AFRL-IF-RS-TR-2003-146
Final Technical Report
June 2003

AGENT BASED ARCHITECTURES FOR DYNAMIC CRISIS MANAGEMENT

University of Rochester

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J387

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-146 has been reviewed and is approved for publication.

APPROVED: Joseph A. Caroli

JOSEPH A. CAROLI
Project Engineer

FOR THE DIRECTOR: James W Cusack

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** JUNE 2003	**3. REPORT TYPE AND DATES COVERED** Final Apr 98 – Oct 02

4. TITLE AND SUBTITLE AGENT BASED ARCHITECTURES FOR DYNAMIC CRISIS MANAGEMENT	**5. FUNDING NUMBERS** C - F30602-98-2-0133 PE - 63760E PR - AGEN TA - T0 WU - 19
6. AUTHOR(S) James Allen and George Ferguson	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Rochester Department of Computer Science Rochester New York 14627	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington Virginia 22203-1714 AFRL/IFSF 525 Brooks Road Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2003-146

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Joseph A. Caroli/IFSF/(315) 330-4205/ Joseph.Caroli@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 Words)***

The focus of this research was on how human users can connect to, participate in, and control agent-based systems. One of the primary accomplishments was the development of a collaborative, dialogue-based problem solving model. The approach incorporates mixed-initiative, dialogue and agent-based technologies. The model was built in a fashion where the human works with an automated intelligent assistant agent that coordinates the interaction with other agents. The model provides a framework for interpreting the user's intentions in the interaction. A dialogue-based system approach proved to be an effective way to support humans when managing agents. Such an approach facilitates intuitive and natural human-computer interaction. The underlying problem solving model supports domain-independent collaboration with a human user. The system was built off of a previously successful dialogue-based mixed-initiative planning system called TRIPS (The Rochester Interactive Planning System). This report discusses the architecture and infrastructure for the system developed. Detail is provided on the model of collaborative problem solving. A number of experiments involving integrated, end-to-end human in the loop agent-based systems are also discussed.

14. SUBJECT TERMS Agent Based Systems, Software Agents, Conversational Systems, Dialogue-Based Systems, Architectures for Intelligent, Cooperative, Distributed, Multimodal Interfaces, Intent Recognition, Interface Agents	**15. NUMBER OF PAGES** 67 **16. PRICE CODE**

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

Table of Contents

List of Figures

List of Tables

1. Introduction

The focus of our research in the CoABS program was to develop new methods for human control of agent-based systems. Our project focused on how humans users can connect to, participate in, and control agent-based systems. This turns out to be an interesting and different research problem for exactly the same reasons that agents are exciting and different from conventional computer systems. In fact, everything that makes agent-based systems "agent-based" affects how humans can interact with, benefit from, and ultimately control such systems.

Agent-based systems represent a paradigm shift in the way people interact with computers. This shift involves moving from seeing the computer as a tool to be used for getting a particular job done, to seeing the computer as an agent that can cooperate to help get things done. As an illustration, consider a typical tool such as a screwdriver. When I see that a screw is loose, I can get out the toolbox, find the screwdriver, apply it directly to the screw (lining up the head of the screw with the end of the screwdriver), twist until I feel the screw is tight enough, then put the screwdriver back in the toolbox and move on to the next job.

This is how people currently use computer systems. If I need to prepare a report, for example, I find Microsoft Word on my machine, launch it, work on the documentation, save my work when I think it's done, and exit Word. My computer is a toolbox full of tools, some of which are easier to use than others but all of which have the following properties:

- They are always available but need to be explicitly invoked to perform their specific task.
- Their use involves directly manipulating the data that they are operating on (indeed, direct-manipulation interfaces are seen as a major improvement over the previous interface technology, which was programming).
- The user must monitor and control the system until the job is done.
- If more than one tool is needed to perform some task or accomplish some goal, integration and application of the various tools is left entirely to the user (some "productivity suites" support rudimentary connections between tools, but these are primitive, *ad hoc*, and usually ignored by users).

Contrast this use of computers with the way people interact with other people to get jobs done. These properties are what distinguish agents from tools, and are representative of the new approach to human-computer interaction represented by humans using agent-based systems:

- Agents are autonomous. That is, they may not be sitting around waiting for us to use them. We may need to find the agents we need to get the job done, just as we would need to put together a team of people with the right set of skills to solve a problem. Further, autonomous agents can and will perform tasks on their own without supervision.

- Agents recognize intention. That is, we do not need to specify every low-level detail of how an agent should perform some task in order to control it effectively. Rather, agents are controlled at a much higher level of abstraction, recognizing the intentions of other agents and determining how best to help (or hinder) those intentions.
- Agents can take initiative. They are not passive objects but are active participants in an ongoing collaborative process involving people and other agents. Agents can take initiative both in communicating information and in performing necessary tasks.
- Agents can delegate responsibility and/or authority to other agents, and can even delegate to human agents. This is crucial to how organizations of human agents are controlled, and is fundamentally different from the use of computers as tools.
- Agents are distributed. This rather obvious point underlies a much more significant aspect of agents, namely that the behavior of agent-based systems is necessarily based on issues of communication and negotiation. These communicative behaviors are both between agents themselves and between agents and people. In fact, given that people are the ones with the problems to solve, it is arguably the human-agent communication and negotiation that dominates any solution to the problem of controlling agent-based systems in practice.

It is precisely these properties of agents that make them interesting, and it is precisely these properties that force us to consider new ways of supporting human control of agent-based systems.

2. Technical Accomplishments

Our work in this project was divided into three main thrusts:

1. Validating and refining our architecture for intelligent user-assistant agents, and continuing development of the implemented infrastructure;
2. Developing a general problem-solving model to support the integration of agents into a collaborative problem-solving process;
3. Development of a series of implemented, end-to-end, human-in-the-loop agent-based systems, both to illustrate the utility of such systems in real-life situations and experimentally validate the models and architectures and drive their further development.

This section will detail our accomplishments in each of the thrusts, although, of course, the research in the various thrusts overlaps to some extent. The final year of the project was funded through the DAML program. The technical goals and accomplishments of this part of the effort are described at the end of this section.

2.1. Architecture and Infrastructure for Collaborative Agent-Based Systems

TRIPS, The Rochester Interactive Planning System (Ferguson, Allen, and Miller, 1996; Allen, Ferguson, and Schubert 1996; Ferguson and Allen, 1998), is a prototype of an intelligent, collaborative assistant that interacts with its human manager using a combination of natural language and graphical displays. The system understands the interaction as a dialogue between it and the human. The dialogue provides the context for interpreting human utterances and actions, and provides the structure for deciding what to do in response. With the human in the loop, they and the system together can solve harder problems faster than either could solve alone.

The TRIPS architecture has always been designed to support agent-oriented components. The infrastructure includes a KQML Facilitator with extensive support for debugging and development, as well as features such as broadcast, status notification, and content-based addressing. TRIPS modules exchange KQML messages via the facilitator, and again the infrastructure provides support for doing this with minimum overhead to the system developers. Specifically, modules can use standard input and standard output for communication, and the TRIPS Launcher sees to it that these streams are connected to the Facilitator when the module is launched. Of course, modules are free to manage their I/O themselves if they wish. This infrastructure has been invaluable in extending TRIPS with new modules with minimum overhead.

2.1.1 An Architecture for More Realistic Agent-Based Conversational Systems

The TRIPS system was designed from the outset to separate the process of interpreting the user's utterances and actions from the performance of problem-solving activities. The former problem was addressed by the Discourse Manager (DM), an agent that responds to user utterances and actions by interpreting them in context in an attempt to recognize what the user intended by them. Once a plausible interpretation is determined, the system typically has some obligation to respond appropriately, such as by answering a question or performing an action. In earlier versions of TRIPS, this was the role of the Problem Solver (PS), an agent that knows how to invoke the various specialized reasoners and other agents in support of user problem-solving activities. For example, it farmed out queries to relevant agents and integrated their responses in preparation for generating a response to the user. In a planning domain such as we have generally explored in previous versions of TRIPS, the Problem Solver coordinated the invocation of planning, routing, and scheduling agents in response to (the Discourse Manager's interpretation of) user utterances. It also maintained the current state of the task, to aid in contextual interpretation and solution generation.

The separation between discourse management and problem solving behavior was an important contribution of the initial TRIPS architecture to the structure of mixed-initiative and conversational systems. However, in working on the TRIPS-CAMPS system (Section 2.3.2), as well as in work on our own "Monroe County/911" domain (Section 2.3.4), we noted some significant shortcomings of the model. The first major problem was that there was really no place in the architecture for external events (i.e., events other than user input utterances and actions) to get "into" the system. In the Monroe County domain, for example, the user is managing a 911-like scenario where new emergencies and problems with ongoing tasks crop up continually. In the past, we had treated these as a special type of "input event," and passed them through the Discourse Manager to the Problem Solver, which knew how to deal with them. Not only was this a kludge, but it really left very little opportunity for the system to make decisions about taking initiative towards the event (e.g., mentioning it, working on it, ignoring it, *etc.*). Second, the Problem Solver was involved in too many things. It performed recognition services to aid the Discourse Manager in interpretation. It invoked specialized reasoners and integrated their responses. It made decisions about how the system should respond to (interpreted) inputs. It maintained state information. And it probably did other things also. Third and finally, the DM–PS architecture placed rigid control of the generation (output) process in the hands of the DM. This precluded a variety of interaction behaviors (such as grounding) that are natural to humans and are essential to supporting realistic mixed-initiative conversation between humans and computers.

To address these shortcomings, we re-designed the TRIPS architecture as shown in Figure 1. In this architecture, the functionality of the conversational system is divided among three autonomous agents:

1. The Interpretation Manager (IM) assumes most of the functions of the old Discourse Manager, except that where the DM also determined how to respond to an utterance, the IM simply outputs plausible interpretations of user utterances and actions.
2. The Behavioral Agent (BA), as its name implies, controls the overall behavior of the system, determining whether and how to respond to events, including interpreted input but also including events reported from other agents and information sources. This is the functionality that was previously divided (rather confusingly) between the DM and the PS.

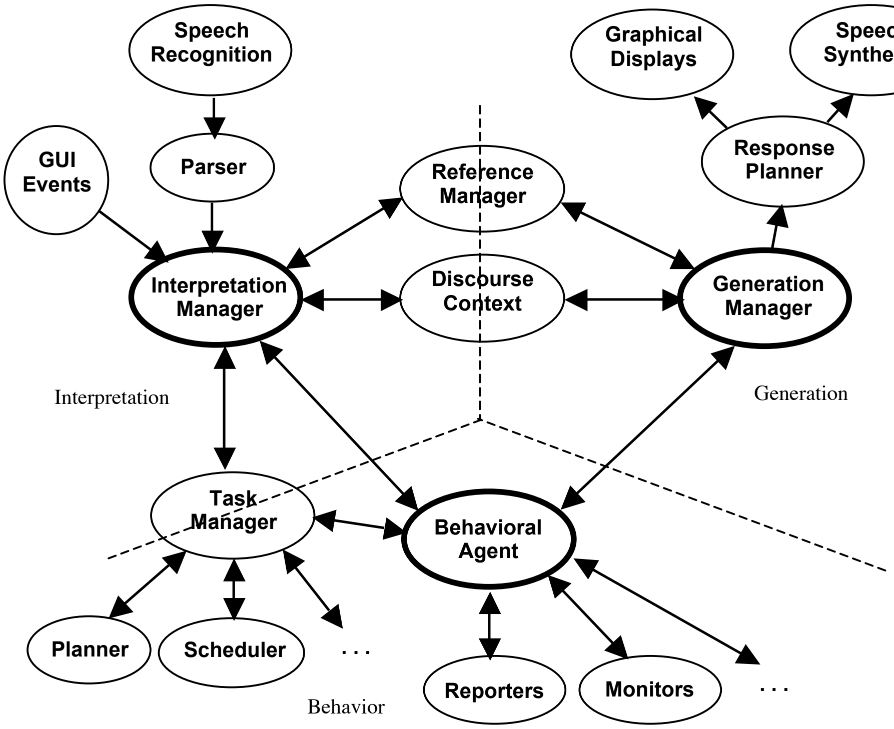


Figure 1: Agent-Based Conversational System Architecture

3. The Generation Manager (GM) now operates autonomously, independent of any ongoing interpretation and behavioral processes. It responds to events from other components, such as obligations arising from user utterances (determined by the IM and maintained in the shared Discourse Context) and requests to perform communicative acts from the Behavioral Agent. The GM is itself implemented as a community of agents operating autonomously to manage user interaction at several levels simultaneously (Stent, 2001).

Of course, these components and the many others that make up the TRIPS system are connected via the TRIPS Facilitator that has proven so useful in the past. While redesigning the TRIPS architecture, we also took the opportunity to reimplement the TRIPS Facilitator in Java. This was really essential for the long-term viability of the system, as well as enhancing portability. It also opens up new possibilities for visualizing the community of agents making up the TRIPS system, by providing hooks with which elaborate displays could be built without changing the core Facilitator code.

The new architecture is described in more detail in (Allen, Ferguson, and Stent, 2001; see Appendix B). The important aspect for our work in CoABS is that the Behavioral Agent is now where it belongs as the “heart” of the system, at least as regards choosing how to respond to events. Since the TRIPS system is generally supposed to be helpful, it will usually respond to user requests immediately. But the new architecture, with problem-solving goals and obligations being maintained by the Behavioral Agent, allows it to, for example, choose to inform the user of a pressing emergency that has just popped up, rather than responding to a less important (to it) question from the user. It can also choose to take initiative on problems as appropriate, requesting that the Generation Manager inform the user at the appropriate time.

This architecture was crucial to the system's performance in domains such as the Hurricane Mitch Honduras scenario developed as part of the MIATA TIE (Section 2.3.3).

2.1.2. Information Sharing Between Agents in TRIPS

One of the important ways agents communicate is by "broadcasting" information to those who might need to know about it. It is unreasonable, on both practical and philosophical grounds, to assume that agents know each other's names and can send each other precisely the messages that they need. Rather, in a large agent-oriented system such as TRIPS, it makes sense to think of (some) agents as knowledge *providers* (sources) and others as knowledge *consumers* (sinks). The agents describe their information needs or capabilities, and the infrastructure arranges for the messages to get from sender to receiver as appropriate for the content of the message. Note that this is a capability that we argued should be added to the CoABS Grid, but the suggestion was not adopted.

In any event, we have supported this type of communication in the TRIPS Facilitator for some time, through what we called a *selective broadcast* capability. Messages without an explicit receiver were candidates for broadcast, but only to those agents that had previously indicated an interest in receiving the sender's messages. This worked, but required knowledge about other agent's names and capabilities that was unworkable in general, as noted above.

We therefore extended the communication model supported by the Facilitator to provide true *advertise* and *subscribe* capabilities. Advertisements are used in a directory-like lookup service to find agents that can perform certain services. We use a simple pattern-based language for specifying advertisements and matching against them, but a more general solution would obviously need to rely on a richer model of the semantics of inter-agent communication (as being pursued, for example, in the DAML program). Subscriptions are similarly based on a pattern-matching language that subsumes and extends our original "listener" model of selective broadcast. We use our subscriptions extensively to structure the flow of information at a semantic level without needing to be concerned with the details of which agents are doing what.

To reiterate the importance of this work, it would be unthinkable to develop a system like TRIPS, with around thirty agents operating autonomously on highly-structured semantic information, without some infrastructural support for the selective broadcast of that information.

2.1.3. Connecting to the CoABS Grid

To make it easier for TRIPS agents to interact with other agents, we developed a general purpose proxy agent that allows a CoABS Grid agent to interact with TRIPS agents via the Facilitator. This agent registers with both the TRIPS Facilitator and the Grid lookup service, and allows an arbitrary Grid component (specified by name) to exchange messages with TRIPS agents. This was used in the TRIPS-MIATA system (Section 2.3.3).

2.1.4. Getting Answers from Agents

One of the problems that cooperating agent systems such as TRIPS have is determining whom to contact in order to get queries answered. Initially in TRIPS, each agent looked after this itself and made queries to other agents as needed. But this became very complex as the number of agents grew, especially when in order to answer some queries, an agent may have to contact several other agents to get the different pieces of information needed to assemble the answer to the query. As well, the set of agents can change dynamically, which requires additional management capabilities to be built into each agent.

To remedy this situation, we developed a new broker for queries, the TRIPS Query Manager. Agents can register with the Query Manager and advertise their capabilities by indicating for what predicates they are able to answer queries. When an agent later makes a query, it is picked up by the Query Manager which then directs it to the appropriate agents. The query manager handles certain logical operators, such as conjunction (AND) and iteration (FOREACH). It can break apart a complex query, use different agents to answer each part, and then combine the results to provide the answer back to the original requester agent. For "universally quantified" queries, such as "What units are available?", the Query Manager can gather results from multiple agents and return the combined list as the answer.

Because the Query Manager's behavior can dynamically change as new agents become available and advertise their capabilities, or as old agents stop providing certain query services, it provides a level of robustness and flexibility that could not be attained if each agent had to hardwire who it needed to ask for information.

It is worth noting that the TRIPS Query Manager is similar to the main feature of the OAA Facilitator, namely matching goals (queries) against agents. In TRIPS, the Facilitator concentrates on facilitating communication between agents, and leaves query brokering to the Query Manager. Note that the TRIPS Facilitator provides content-based addressing (subscription) that allows the Query Manager to handle queries without the querying agents needing to know whom to ask for the answer. The OAA Facilitator is somewhat more complex at this point, since it can be "programmed" with Prolog rules that allow it to do inference in order to answer queries. Of course, the Query Manager, being a separate agent itself, can equally well be programmed (and is currently, for the extended capabilities described above). It would not be hard to integrate a Prolog engine into the Query Manager if that was the most convenient way to specify its behavior (we're not sure that it is, mind you).

The Query Manager is fully implemented and in use in current versions of the TRIPS system. Future development will involve enhancing the advertisement model beyond simply naming predicates and performing some forms of query optimization to improve performance.

2.2. A Model of Collaborative Problem-Solving

The work described in the previous section is the foundation on which conversational, collaborative assistants can be built. In the section we describe the work we have done specifically to address the needs of collaborative problem-solving in agent-based systems.

2.2.1. Requirements for Collaborative Problem-Solving

In previous versions of TRIPS, there was a tight linking between the abstract problem solving (generic across all task-oriented interactions) and the specific plan reasoning required in order to build deployment plans. To move towards more generic control of agent-based systems, we have had to separate out the different components. Specifically, there are three different capabilities required of an agent-based collaborative problem-solving system:

1. Managing the problem solving state: what goals and preferences, what solutions, and what different scenarios are being considered.
2. Task modeling: using a model of the human's task in order to recognize intention, take initiative when desired, and plan the presentation of relevant information to enhance the human's situation awareness.
3. Managing the tasking of agents: what agents are available and what capabilities do they provide, how are results coordinated between the agents over the longer term.

2.2.2. Collaborative Problem Solving Management (An Initial Approach)

The key to effective collaborative problem-solving from the system's perspective is that human high-level goals must be translated into specific taskings for the set of available agents. In our first attempt, this was all done by the TRIPS Problem Solving Manager (PSM). In our model, the PSM was extended to support the following operations: (1) plan recognition to support the Discourse Manager in choosing among multiple interpretations (intention recognition); (2) reasoning about how to use the available agents in the current problem-solving context (in order to be most helpful); and (3) the actual management of the interaction between the PSM and the sub-agents, including formulating messages and gathering replies.

A preliminary version of the new PSM was in place in the CoABS Science Fair demonstration of the TRIPS-CAMPS system in Washington in October, 1999. As described in Section 2.3.2, that system involved TRIPS agents working with the CAMPS-MP Air Mobility Command mission planner and an agent that plays the role of the AMC "Barrel". The Barrel interacts with the actual units in the field in order to know what assets are available, and allocates specific assets to requirements when requested.

The new PSM was able to make plans to interact with these two agents in order to accomplish goals. For instance, after a question about resource needs, the system answers "4 C141s and one C5 will be needed". In processing the subsequent question "Where can we get them", the problem solver plans to locate 4 C141s and 1 C5. It knows that the Barrel agent can only answer queries about one plane type at a time, so it plans two calls, one for C141s and one for C5s, and then combines the results to produce an answer for the user. In other cases, the problem solving must call the Barrel agent to find resources, and then call the scheduler with those resources to produce a schedule.

2.2.3. An Agent-Based Model of Collaborative Problem-Solving

As described in the previous section, we ultimately redesigned the TRIPS system, with the goal of providing a general-purpose architecture for building conversational assistants

Abstract Problem-Solving Model Element	Example Instantiations in Monroe County Domain
Objectives	Treating heart attack victim at Marketplace Mall; Repairing a downed electrical wire; Locating a crew; Plowing a road
Solutions	Send ambulance 61 to Marketplace Mall to get the victim and transport them to Strong Hospital; Dispatch a crew from sector 3 to assess and repair the downed power line
Resources	Vehicles (ambulances, repair trucks, plows, helicopters); Crews (medical, repair, plow); Roads; Fuel, Time
Situations	Location of vehicles; Availability of crews; Status of roads and bridge; Severity of damage; Type of medical emergencies

Table 1: Instantiation of Abstract Problem-Solving Model Objects in Monroe County 911 Domain

(Figure 1). The major changes involve a cleaner separation between the language and dialogue components of the system and the task- and domain-specific reasoners that provide the system's knowledge-based capabilities. This separation enhances portability to new domains, and allows independent development of the different levels of the system.

The key to making this separation work is the specification of an interface language between the levels. In our case, this is an abstract model of problem-solving that captures the generalities of collaborative problem-solving behavior across a variety of domains (Allen, Blaylock, and Ferguson, 2002; see Appendix C). The model defines several classes of entities that are used in problem-solving, such as objectives, solutions, resources, and situations. There are then a variety of operations that can be performed on these objects, such as creating an objective, extending a solution, identifying a situation, and so on. In general, these operations are fundamentally collaborative—they cannot be achieved by one party alone. So the model allows one participant to initiate a problem-solving act, say of extending a solution. The other participant must then complete the act, say by accepting the suggestion. They can also reject, abort, or clarify the problem-solving act initiated by the other participant. Table 1 gives some examples of how the various objects of the model are instantiated in our Monroe County 911 domain (Section 2.3.4). As is evident from the table, even in this simplified domain, the range of possibilities for collaborative problem solving is quite extensive, as well as being representative of other problem-solving domains. However, a model this rich is absolutely essential in order to specify in any principled way the behavior of the collaborative system.

A key aspect of our development, which separates our work from more language-centric analyses, is that we are at all times concerned with actually grounding the problem-solving in the system's capabilities (its own and those of other agents that it can locate and use). It is not enough for us to simply categorize a statement as “extending a solution.” We need to figure out how that recognition might be done by the Task Manager in a real context with real plans under consideration, then figure out how the Behavioral Agent can coordinate the invocation of planners and schedulers (or whatever agents are appropriate to the task and domain) to actually compute the new solutions, analyze them, and produce responses to the user.

Compared to more language-oriented efforts, a significant part of the problem we are solving involves getting the system to actually do something intelligent. Most other collaborative systems with which we are familiar perform relatively straightforward “back-end” tasks, such as looking up flight schedules or locating Chinese restaurants. In these domains, it is simple to determine how what the user said contributes the problem-solving. Fore example, if they said something including “from Chicago,” then fill in the “from” slot of the airline flight query template with the value “Chicago.” Then, in these simple domains, it is trivial to specify how the system should perform the task. In this example, send the query to the database and wait for the answer.

But in more realistic domains such as we have been considering in CoABS, the task that the human-agent team is solving is more complicated, making it harder to recognize what the user might be doing (since they could be doing more things), as well as harder to “solve” the problems (generate solutions). Further, in our CoABS work we have concentrated on cases where the TRIPS system by itself cannot solve the problems with its own reasoning components, but must rely in whole or in part with external agents with whom it must communicate in order to produce solutions, analyses, or presentations. In cases where we don’t have the external reasoners, we can still make progress by designing our own reasoners to meet a general API, then force the rest of the TRIPS system to interact with them as black boxes.

2.2.4. TRIPS Task Manager Agent

Eventually, our new model of collaborative problem solving led to the development of a new component of the TRIPS architecture, the Task Manager (TM). This agent encapsulates the system’s task- and domain- specific knowledge, and supports both recognition and execution, as we will describe in this section.

Previously in the TRIPS architecture, a line was drawn between *interpretation* and *behavior*. That is, one set of agents and components, centered around the Interpretation Manager, were involved in interpreting the user’s utterances and actions. Meanwhile, the system’s overall behavior was controlled by the Behavioral Agent, which managed a suite of agents to actually perform the system’s parts of the collaborative task. The interface between them is in terms of the abstract problem-solving model described in the previous section. This was a crucial insight and, once implemented, cleaned up many aspects of the system. In particular, it provided, perhaps for the first time in mixed-initiative systems, a clear place where the initiative-taking behavior of the system was specified and controlled.

However, this splitting of responsibilities started to lead to some duplication of effort. Or worse, in places where the effort was not duplicated, similar reasoning tasks were being performed differently, with the result that an interpretation could be produced that could not then be executed. While this is not necessarily wrong, it shouldn't happen simply because of bad system design.

It became clear that there is a core set of knowledge about the task and domain at hand that is *shared* between interpretation and execution. For example, the interpretation components need to know "does it make sense that the user would be doing *X*?" while the execution (behavior) side needs to decide to what to do in response to the user doing *X*. In a collaborative setting, these tasks are clearly complementary. If we can figure out that probably the user is doing *X*, then we can know something about why they're doing it (for example, a higher-level goal that they are pursuing), and from that infer what we should do (as part of a shared plan to achieve the higher-level goal, for example).

To encapsulate the system's task- and domain-specific knowledge, we designed the Task Manager component. In the TRIPS architecture, this is the only components that straddles the boundary between interpretation and behavior. The Task Manager is based on a hierarchical plan representation of user activities for the task at hand, and by a declarative mapping from elements of the abstract problem-solving model to objects in the domain. As this is just the first version of the Task Manager, these representations are still evolving. But as an example, in the TRIPS Monroe 911 domain, an task model activity might be responding to an emergency. This can be decomposed into several steps, some of which are prior to responding, such as identifying the location and severity of the emergency, others of which take place during the response, such as checking that the assigned vehicles make it on site in time, and so on. Similarly, the domain model in TRIPS Monroe 911 would indicate that responding to emergencies is one type of goal, ambulances and repair crews are among the resources available for use in solving problems, and that situations are defined by such attributes as location of objects on the Monroe County map and severity of injuries to people.

Based on this knowledge, the Task Manager can handle two types of requests. From the Interpretation Manager, it can be asked to INTERPRET a set of possible user acts in a given context. It responds with a set of interpretations, which each include both the meta-collaborative problem-solving act that the Task Manager thinks accounts for the user act, and a recognition rating, indicating how likely the Task Manager thinks the interpretation is. From the Behavioral Agent, the Task manager can be asked to PERFORM a collaborative problem-solving act. The response depends on what act was performed. Sometimes there is content to be communicated to the user, sometimes it is just that the status of the task has been updated.

A brief example will clarify the way the Task Manager supports both interpretation and execution. Internal details of the representation will be grossly simplified for this report (the actual knowledge representation used to exchange information between components in TRIPS is very rich and quite expressive). Let's say that, in the Monroe 911 domain, the user asks "Can I use a helicopter?" during the discussion of responding to some emergency. The Interpretation Manager will produce (at least) the following two interpretations: either this could be a simple yes-no question about the user/system's abilities in general, or it is a request to evaluate a proposal to do something with a helicopter. The Task Manager will rate the second interpretation higher (assuming that the person probably wouldn't ask such a basic yes-no question in the 911 domain). It will also return that this is an INITIATE (a meta-collaborative problem-solving act) of a C-EVALUATE-

FUTURE-ACTION (a collaborative problem-solving act, which involves both the user and the system) about the use of a helicopter. This interpretation, once chosen by interpretation, is broadcast as the system's understanding of what the user has just done.

This is picked up by the Behavioral Agent, which has a standing goal of being cooperative. In this case, that amounts to requesting that the Task Manager PERFORM an EVALUATE-FUTURE-ACTION act (a simple, non-collaborative, problem-solving act). The Task Manager responds with the evaluation, which it obtains from the various task- and domain-specific reasoners at its disposal (perhaps a planner and a resource database, in this example). For example, it might answer that this would be a good thing to do considering the options, and perhaps further communicating the changes in the current operations needed to accommodate the new helicopter mission. These are communicated to the user by the Behavioral Agent via the Generation Manager in an attempt to COMPLETE the collaborative act initiated by the user (assuming the BA didn't have anything more important to do by the time the TM was done). Once the user grounds (acknowledges) the system's response, the collaborative problem-solving act is in fact completed.

A preliminary prototype of the Task Manager is in use in the current TRIPS system. There is plenty of future work here, including completing the specification of the Monroe County task and domain models, and refining the collaborative problem-solving model to properly account for a broader range of collaborative activities that might arise between users and agent-based systems.

2.3. Integrated, End-to-end, Human-in-the-loop Agent-Based Systems

2.3.1. TRIPS-Pacifica

At the start of this project, we were working on the TRIPS system to improve its generality and portability to new domains, especially in its capabilities to connect to different agents to perform its back-end reasoning. This work was performed using the "Pacifica" domain originally developed in the DARPA Planning Initiative (ARPI) program. This is a simple crisis logistics or "NEO" domain where the human manager needs the system's assistance to plan the evacuation of the residents of the island of Pacifica ahead of an approaching hurricane. This system provided the baseline for the subsequent systems described below.

During our CoABS work, we delivered a standalone demonstration version of TRIPS-Pacifica to AFRL. The amount of work that this required should not be underestimated.

2.3.2. TRIPS-CAMPS

Fairly early in the project, it was suggested that we had not proven the case for TRIPS as an agent-based system itself, apparently since we had developed all the agents ourselves. We therefore began work with BBN towards integrating the CAMPS-MP airlift mission planner into a new version of TRIPS.

The CAMPS-MP mission planner is designed to support planners at the Air Mobility Command (AMC) in scheduling crews, cargo, and planes for Air Force missions. Rather than adapt CAMPS-MP to another domain, we decided instead to develop an airlift mission planner domain for TRIPS. This would illustrate again the relative ease with which

TRIPS components can be ported to new domains, and would provide us with another opportunity to improve those aspects of TRIPS that remain difficult to port. It also served as a key demonstration that TRIPS is capable of interacting with external agents that were not designed with a dialogue system in mind. In performing this experiment, we discovered what aspects of external systems are amenable to the incremental style of processing in TRIPS, and identified requirements on external modules that would allow us to enhance the level of integration between human agents and these systems.

Our integration effort had the following objectives:

- Demonstrate that the TRIPS system could interoperate with components not developed at Rochester.
- Demonstrate that the TRIPS system could be adapted to perform a task other than the NEO scenario on which we had concentrated previously.
- Demonstrate the effectiveness of mixed-initiative interaction in the performance of a realistic task that is relevant to DARPA and its customers.
- Start laying the groundwork for the MIATA TIE effort, that would involve both TRIPS and CAMPS along with additional components.

In what follows, we will address briefly how each of these objectives was met.

First, we validated the design and implementation of the TRIPS infrastructure by connecting the CAMPS MP-Agent from BBN into the system. The TRIPS Facilitator-based infrastructure (described previously) was invaluable in making this connection almost painlessly. The CAMPS MP-Agent from BBN is an encapsulation of the CAMPS Mission Planner (MP) within a KQML message-passing shell. Since the MP-Agent was already prepared to communicate via a KQML API over socket-based connections, the integration with the TRIPS Facilitator was very straightforward. We had little trouble integrating the MP-Agent into the overall TRIPS configuration, so that it could be started and stopped conveniently with the other components of the TRIPS system.

The second aspect of the TRIPS-CAMPS integration involved adapting TRIPS to help the human mission planner with their task. First, we had to determine the scope of that task for purposes of the integration experiment. In collaboration with BBN, we developed a simple model of the mission planning process in which the human mission planner uses the CAMPS Mission Planner as a tool, providing it with requirements, assets, and additional resources and constraints. The CAMPS MP-Agent API was extended to support this model through the addition of several KQML messages.

At this point we decided to complicate the picture by introducing another agent in the integration experiment. In the mission planning task, the human planner must request resources from a central scheduling authority known as the “Barrel.” With BBN, we developed a BARREL agent that could answer queries about resource availability via a KQML interface (a more realistic barrel agent would also manage resource allocation).

The integration effort ,while initially *ad hoc*, was ultimately based on the concrete (albeit simplified) model of the task. At the time, task- and domain-level knowledge in TRIPS was managed by the Problem Solver, which mapped the relatively domain-independent output of the Discourse Manager into specific task- and domain-level actions, and coordinated the invocation of the specialized reasoners used in TRIPS. In this case, the reasoners involved were the CAMPS MP-Agent and the BARREL agent, rather than, for example, the temporal planner, router, and scheduler used in the Pacifica domain. We developed a new version of the Problem Solver to operate in the airlift mission planning domain and coordinate the interaction with the CAMPS MP-Agent (*e.g.*, to add requirements or assets, to request a schedule, or to request analysis of various problems that could arise) and with the BARREL (mostly to determine asset availability for use in scheduling requirements).

The third objective, and one of the most important in our minds, was to show the utility of natural mixed-initiative interaction in a realistic task. Despite our simple model of the mission planner's task, and the limitations of TRIPS, CAMPS, and our BARREL agent, the demonstration of the resulting system was very effective. While almost everything we did conversationally in the demonstration could have been done with the CAMPS-MP graphical user interface, it would have been more awkward (unless the user was a CAMPS-MP expert, which would require extensive training). Furthermore, in some cases the TRIPS-CAMPS system, using its task model, could perform several actions and combine their results in order to meet the human planner's intentions. It is these "compound operations" that are difficult to perform with a GUI, but that are crucial to performing real-world tasks effectively.

Finally, this integration effort was used as a "warmup" for the larger-scale MIATA TIE which followed (see next section). But the work we did on the TRIPS-CAMPS system was crucial to both TRIPS' and CAMPS' successful performance in the MIATA demonstration's even large agent-based system of systems. A paper documenting our integration efforts was presented as a poster at the ICMAS-2000 conference (Burstein, Ferguson, and Allen, 2000; see Appendix A).

During the project, we delivered a standalone version of the TRIPS-CAMPS system to AFRL.

2.3.3. TRIPS-MIATA

The next implemented system that we developed as part of our CoABS effort was for the Mixed-Initiative Agent Team Administration (MIATA) Technology Integration Experiment, a large-scale, live demonstration involving agents and systems from many different institutions.

Work on the scenario for the MIATA demonstration had been ongoing since the start of the CoABS program. From the outset, we felt that a natural disaster scenario would include all the elements of the simpler NEO scenarios used previously, while allowing us to broaden both the scope of the operation (services involved, tasks to be performed, resources to be used, *etc.*) and the range of agents involved (command and control, scheduling, execution, monitoring, reporting, *etc.*). Primarily due to the efforts of BBN, we had extensive data on the real-life response to Hurricane Mitch in Honduras. One challenge was that we needed to show in a few minutes (say ten) an operation that took weeks in real life. We chose to do certain things in real time and use the "fast time" capabilities of the underlying simulator to quickly move through routine (though still interesting) tasks for purposes of demonstration.

The specification of the final scenario proceeded in parallel with the determination of which systems would be playing which roles in the demonstration. Our efforts focused on the role to be played by TRIPS, and in particular which roles would benefit from the kind of human-in-the-loop mixed-initiative interaction that TRIPS supports so well. With BBN, we decided that the primary role would be an intelligent assistant for the Commander of the Joint Task Force in charge of the operation. The TRIPS system would assist him or her in putting together the team of agents making up the JTF, tasking them as appropriate for the situation in Honduras, and then monitoring their operation and making command decisions to resolve problems as the mission progressed. We originally envisaged a second TRIPS system supporting a human user in a different role: the dispatcher managing the delivery of supplies from their arrival at the airport to the hardest-hit Honduran towns. This person would be under the command of the J3 (operations), and problems encountered during the mission would percolate up the command chain depending on the action (and authority) necessary to resolve them. A third role for TRIPS in the MIATA scenario was as the assistant to the AMC mission planner responsible for getting the supplies to Honduras in the first place, a role we could support using the TRIPS-CAMPS system developed earlier in the program.

As the specification of roles became clearer, we completed the instantiation of the new TRIPS architecture for the CJTF role. As noted in Section 2.2.1, we had previously redesigned TRIPS to better support interaction with external agents and response to external events, both of which are pervasive in the Hurricane Mitch scenario and the CJTF role in particular. The majority of the work here, as expected, was in specifying the task that the commander was performing in sufficient detail for TRIPS to be able to assist in it. This required task-specific versions of the Task Manager and Behavioral Agent components of TRIPS. The task specification we had for this role was not very detailed, so the resulting components (and system) were not very robust (in the sense of being able to support many aspects of the task), but they were a “first cut” at validating the new TRIPS architecture. Certainly, it would have been impossible, or at least very ugly, to support the type of mixed-initiative reporting required by the Hurricane Mitch scenario using our old architecture. It was decided rather late in the game to not use a TRIPS-human system for the dispatcher task. We had done some work on this, and the new architecture was designed to support it, but there were other agents brought into the MIATA club that could handle dispatching. This type of task is, however, at the core of our Monroe County 911 domain (see next section).

We planned from the outset to use the CoABS Grid in the MIATA TIE. We decided not to route all message traffic between TRIPS components via the Grid, because we saw the Grid as a connector of systems, rather than a replacement for existing communication schemes. For this, we used our Grid proxy agent described in Section 2.1.3. As a proof of concept, we implemented two simple databases as Grid components, then connected them to TRIPS using our proxy agent. The two data sources are used transparently by the TRIPS Behavioral Agent to answer queries during the demonstration.

The MIATA demonstration scenario is driven by the MapleSim simulator from CMU. This system simulates the transit of Hurricane Mitch across Honduras using actual data, simulates the effect of the storm on roads, bridges, and towns, and simulates the operation of trucks delivering supplies throughout the country. MapleSim communicates via a

KQML interface, making it straightforward to integrate with other components. In order to test our components (and in support of the "truck dispatcher" role originally planned for the demonstration), we developed agents capable of driving the simulated trucks in response to higher-level orders, as well as a visualization tool based on the TRIPS Map Viewer and using the MapleSim KQML API. Although they did not figure in the initial MIATA demonstration, they will be used extensively in our Monroe County 911 scenario.

Finally, to support the use of the CIRL scheduler in a dynamic situation (which the Hurricane Mitch scenario certainly is), we implemented a version of the TRIPS Router that could find ground routes for trucks using the road network specified by the CMU simulator. This wasn't particularly hard—we had a general-purpose scheduler based on Dijkstra's algorithm already in TRIPS. An interesting issue arose when we noticed that the representation of the road network used by the simulator involved on the order of ten thousand points. The vast majority of these were not relevant to routing decisions (did not involve intersection of roads) but were merely present in the GIS dump that made up the road network specification. To make the routing algorithm practical, we made the router dynamically create a routing subnetwork containing all the relevant points (about one thousand), use that in computing routes, then expand the resulting route to contain a specification suitable for the simulator.

Of course, a key issue for large demonstrations is coordinating the interactions between agents and components from several different places. We spent plenty of time refining interfaces and debugging messages. As before, the TRIPS infrastructure made this much easier for components that worked within it. To support the needs of other TIE members, we developed components that allow multiple TRIPS components to run within a single Lisp image or Java virtual machine. This is completely transparent to the components, which are built using infrastructure we have developed over the years, and is absolutely necessary for platforms with less CPU horsepower or memory. Performance suffers somewhat on these platforms, of course, but the system runs flawlessly.

The August, 2000, MIATA demonstration involved the following groups:

- University of Rochester: JTF Commander's Assistant (twenty or so components supporting spoken language interaction, Facilitator architecture, Grid interconnectivity)
- BBN: JTF Agents (multiple agents, internal KQML messaging, GUIs); CAMPS-MP; JADE; Grid proxy
- SRI: J2 (Intel) agent
- CMU: MapleSim simulator
- Kestrel: Glue-code generator
- OGI: Scheduler to schedule delivery of incoming supplies to towns

In addition, several players were added at the last minute, such as the Grid Visualization project.

The first hurdle we overcame to make the demo a reality was that not all the software was ready to play. Some stuff didn't work as expected. Some people didn't know how to make their stuff work in the demonstration environment. Some people never delivered their stuff. We worked through these problems in the lead-up to the demo.

Next, we needed enough cycles to run everything. Not much thought had been given to local arrangements. We shipped in our workhorse Ultra60 Sun workstation and it was really the computational backbone of the demo. In addition to running all the TRIPS components making up the JTF Commander's Assistant, it ran the SRI J2 agent, BBN's CAMPS-MP, and the Grid service (the latter of which consumed huge amounts of resources).

Next we had to interconnect all the other pieces. In the end there were at least five different systems running three different operating systems (Solaris, Windows, Linux), all connected by an ad hoc LAN that we administered. Some people didn't know how to make their software work on this LAN, so we looked after that also.

The entire system was setup before the demo, then broken down and moved to the main conference room the morning of the demo.

The demonstration itself went very smoothly. The systems performed flawlessly. While we had a "script" in order to structure the presentation and make sure that certain elements were shown (in order to illustrate important points about mixed-initiative interaction in human-computer agent teams), it is important to note that the MIATA demo is running live. All the agents are reacting to events that occur in the simulated version of Honduras provided by MapleSim. This is very different from, e.g., the earlier NEO TIE, where all the events are part of the script. The MIATA demo is inherently somewhat unpredictable, and in fact it takes some effort to make the systems "perform on cue" for demonstration purposes. We believe that this makes for a more compelling demonstration, however, which is why we have been building and demonstrating end-to-end systems for several years.

In any event, in the course of the demo, the JTF Commander put together his team, tasked them out, then waited for reports and problems to arise. The J2, J3, and J4 agents accepted their taskings and performed the following tasks. The J2 (Intel) coordinated the use of truck and helicopter agents to determine of the extent of damage and formulate needs requirements. The J4 (Logistics) worked to meet those requirements by arranging for delivery of supplies (interacting with JADE and CAMPS-MP systems). The J3 (Ops) then worked to get the supplies delivered as they arrived in-country. The system could be seen working through either the Grid Visualization tool, or by observing the agents working in the simulation, or by looking at the GUIs of the various agents in cases where they had one. Underneath it all, MapleSim agents were actually driving or flying around the country, and reporting their findings back to their superiors.

For the second MIATA demonstration, six months later, our objective was to build on the successful execution of the previous demonstration, but add features and complexity that highlighted the range of issues involved in managing mixed-initiative human-agent teams and their possible technological solutions.

The main changes for us involved the fact that we wanted to do a better job of illustrating the flow of initiative between the levels of agents in the system. The basic model is that authority flows down from the top, as agents are tasked by their superiors. They may work on these tasks or delegate them to other agents. As they are working their tasks, problems (or more generally, unanticipated situations) may arise. An agent has the choice of taking initiative and trying to resolve the problem (deal with the situation) itself, or of yielding the initiative and passing the problem back to its superior. Thus problems (or more generally reports, including positive reports) flow upstream from subordinate agents to their superiors.

The major issue for designing agents that can participate in these mixed-initiative environments is how to specify the situations in which they should take initiative and how to specify those in which they should yield it (and to whom). The transfer of initiative has been studied extensively in the area of conversational systems, where so-called "dialogue initiative" controls which participant is expected to speak next. But the initiative to solve problems corresponds more closely to so-called "task initiative," which has been much less extensively studied in the mixed-initiative literature. In addition to the behavior of the individual agents, there is the meta-question of specifying (or analyzing) the behavior of the multi-agent system (organization) as a whole. This is very difficult problem, and while we do not yet know how to address it, the MIATA demonstration is intended to at least illustrate the issue in a realistic scenario.

Those were our goals for the January, 2001, demonstration. As with the previous demo, there was some significant engineering work to pull all the pieces together and have them interoperate. The number of machines involved increased (to eight or nine), and we orchestrated the displays onto three screens. We will not dwell on these details—suffice to say that as previously, some people's software wasn't ready to run, and some people's software never ran properly.

Technically, the demonstration was perfect. The systems interacted exactly as we had hoped, with taskings flowing down from the JTF Commander to the agents representing the JTF staff and thence to various domain agents like truck drivers and helicopter pilots. As anomalous situations were encountered, such as the Intel group not being able to reach some towns to perform needs assessment because of bridges being out, the reports flowed upstream , ultimately to the Commander who was able to make an executive decision such as reassigning some assets from Ops to Intel temporarily.

For our specific part of the demonstration, namely the JTF Commander's Assistant, both the spoken language processing and the back-end interaction with the JTF Staff agents worked perfectly. As noted above, the models we used for this version of TRIPS were not the most robust, since we had neither detailed data on the language that such a commander might use, nor, more significantly, any detailed model of how they might solve the problems they encounter. Nonetheless, focusing on the language used in the demo and working with an API for the Commander's staff agents, we feel that the Commander's role in the MIATA demonstration made a convincing case for the kind of conversational assistant that we think is essential to allow users to interact with and control agent-based systems.

We delivered a standalone version of the TRIPS-MIATA system to BBN for use during development of the MIATA TIE. After the second demonstration, we delivered a video of the live demonstration to DARPA and BBN.

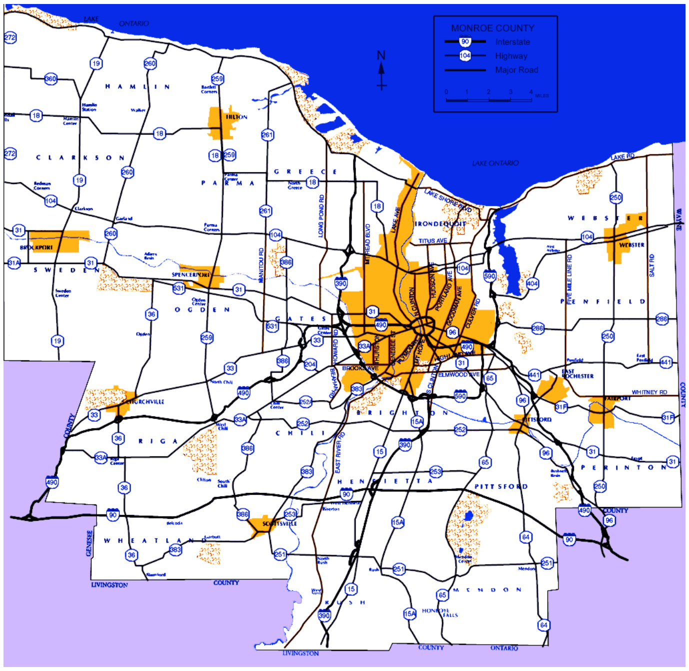


Figure 2: Monroe County 911 Sample Map

2.3.4. TRIPS-Monroe/911

During our CoABS project, we also began development of a new domain in which TRIPS can operate. This work again validates our claims of (relative) domain-independence of the existing version of TRIPS and allows us to refine those aspects of TRIPS that are difficult to adapt, making subsequent ports much simpler. A screenshot of the map for this domain, based on a map of Monroe County, New York, is shown in Figure 2. The real-world map is obviously significantly more complex than the previous Pacifica domain, and leads to a variety of difficult problems in such areas as linguistic reference to objects and places in the domain, computational complexity of tasks such as route-finding, and significantly more realistic and more difficult collaborative problem-solving behavior overall. We will, in addition, be extending the task being performed collaboratively to include management of multiple sets of resources (for example, rescue crews and repair crews in an ice storm), possibly being managed by multiple people, and ongoing execution and monitoring based on reports from agents in the field.

In order to bootstrap this work, we collected six hours of sessions in which people collaborated to build, describe, evaluate and modify plans. The experiments were designed to elicit extensive interactions to develop brief plans, involve much answering queries (drill down), and involve extensive plan modification. This data will serve as a resource for several years to come in developing more natural automatic problem solving agents that complement this human problem-solving rather than interfere with it

After the MIATA demonstration, we began trying to apply the progress we had made in the MIATA domain to our base experimental system in this new Monroe County 911 domain. The MIATA domain is not particularly suitable as a testbed for our development of mixed-initiative agent-based systems with humans in the loop. The main reason for this is that we really have almost no significant model of the task the JTF Commander is performing during the management of the crisis. It is simply too complex, in a domain that we do not understand well enough. The role that the Commander plays in the MIATA scenario is primarily driven by the needs of the demonstration, which are unfortunately not quite in line with our longer-term research plan.

The problem is that if we can't model the task the user is performing, then we really can't build a collaborative assistant to assist them with it. Specifically, while we can build language systems that superficially "understand" the way the Commander talks, we cannot perform the crucial next step of interpreting their statements (and actions) in context to recognize the intention behind what they said (or did). Without this, it is impossible to build an assistant agent that, for example, adopts the user's goals as its own and tries to achieve them, or recognizes obstacles to the user's goals and mentions them or tries to remove them. Intention recognition is in fact critical to agent-agent interaction—to see this, think about how much you use your sense of what someone is trying to do when you interact with them.

Additionally, the MIATA domain is too complex for us to build reasoners that can produce solutions to problems faced by the Commander, even if we could recognize what those were. The goal of our work in MIATA is not to build planners, schedulers, and knowledge bases capable of reasoning about large-scale relief operations. But such components are necessary in order to build intelligent systems that can collaborate with people. So if we want to study human control of agent-based systems, we need to work in a domain where these systems can contribute to the collaborative problem-solving.

The Monroe County 911 domain is intended to be just such a domain. It is obviously a simplification relative to something like the JTF Commander's role in the MIATA scenario. On the other hand, it is significantly more complex than the role played by the user in our previous Pacifica NEO domain. The use of a real geographical setting (indeed, one that we and most of our users know intimately) provides a variety of challenges from language to reasoning. The specification of the 911 dispatcher's task as one of monitoring problems in a dynamic world (provided by a simulator) and incrementally developing, executing, and refining solutions pushes the state of the art at every level in the system. Our goal is to make the domain realistic enough that the lessons learned can be generalized to more realistic domains and bigger and harder tasks, while still giving us the control we need to simplify aspects in order to accommodate our current partial understanding of the technical problems and their solutions.

This work began under CoABS and is continuing under other funding.

2.4. Temporal Ontologies for the Semantic Web

For the final year of the project, our focus changed in order to address the goals outlined in a SOW for a supplement funded through the DAML program. The issue there is that as the web becomes populated by agents and services rather than pages and CGI scripts, the interactions between them will become significantly more complicated. In particular, agents will need to represent and reason about time, both in order to adequately describe dynamic worlds and to control and coordinate their own behavior. While there has been extensive work on temporal reasoning in the past, most work on temporal ontologies to date glossed the complexities of time. Our project focused on developing ontologies and representations for describing dynamic temporal aspects of the world and developing some useful DAML temporal ontologies. Specifically, the goals were:

- Survey prior work on temporal ontologies in DAML
- Participate in the development of an "upper-level" ontology of time
- Develop a practical temporal ontology suitable for use in marking up web data

We will comment briefly on each of these.

First, we completed a survey of the existing work on temporal ontologies within the DAML program as part of development of some general temporal ontologies. While there are some very extensive ontologies, such as the CYC ontology, it is overly complex for most applications, and the temporal ontology is not easily separated from the rest of the CYC ontology, which creates a high hurdle for most applications, where the needs for expressing temporal information are more modest.

Second, we participated in a multi-site collaboration led by Jerry Hobbs on defining an "upper-level" temporal ontology that could be shared among all the different researchers in the DAML program. We played a key role in formulating and debugging the core axioms so that different theories of time (e.g., point-based, interval-based, continuous, discrete, and so on) were not excluded. A particular focus was the development of a set of additional axioms that extend the core axioms to produce Interval Temporal Logic. We developed automated and semi-automated tools for representing this ontology in DAML, using an encoding of first-order logic in RDF developed by Drew McDermott and others.

Third, we have made significant progress on defining a simple but useful temporal ontology that can be used to capture the everyday temporal information in documents and data. It consists of a small set of primitive relations that generalize Interval Temporal Logic (Allen, 1984; Allen and Ferguson, 1997) by allowing optional metric information. For instance, there is a relation BEFORE that states that one time is before another, with an optional temporal duration that constrains the allowable time between the two times. Thus, a phrase like "X was 5 hours before Y" would be captured quite directly by a BEFORE relation between X and Y with a separation duration of 5 hours. Interestingly, if we allow the duration constrain to be 0, then this relation corresponds to the MEETS relations in Interval Temporal Logic. Because of this collapsing, instead of requiring 13 primitive relations, the proposed ontology only requires five. In the future we will be evaluating the use of this ontology for marking up real-life data, and continuing to develop it.

Finally, we developed and maintain the DAML-Time web site (http://www.cs.rochester.edu/~ferguson/daml/) on which we will maintain different temporal ontologies, including the "upper-level" ontology, our extensions for ITL, and ultimately the new "user-friendly" temporal ontology we are developing. This work is ongoing as part of a new project funded by the DAML program.

3. Conclusion

In the course of this project, we have refined and developed our architecture for agent-based collaborative systems, clarifying the responsibilities of the different agents and their interfaces. We have developed a model of collaborative problem solving that not only provides a framework for interpreting the user's intentions in the interaction, but is also used as the communication language between the agents that comprise our system. And we have developed and demonstrated a series of implemented, integrated, end-to-end, human-in-the-loop systems in realistic domains. These systems both validate the underlying models and components, and illustrate vividly how necessary such systems are to enable human control of agent-based systems.

4. References

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence* **23,** pp. 123–154.

Allen, J., N. Blaylock, and G. Ferguson (2002). A Problem Solving Model for Collaborative Agents. *Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, Bologna, Italy, July 31–August 2.

Allen, J., D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent (2001). Towards Conversational Human-Computer Interaction. *AI Magazine* **22** (4), pages 27–38.

Allen, J. F. and G. Ferguson (1997). Actions and events in interval temporal logic. Oliveiro Stock (ed.), *Spatial and Temporal Reasoning*, Kluwer Academic Publishers, pp. 205–245.

Allen, J., and G. Ferguson (2002). Human-Machine Collaborative Planning. *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, Houston, TX, October 27–29.

Allen, J. F., G. Ferguson, and L. K. Schubert (1996). Planning in Complex Worlds via Mixed-Initiative Interaction. *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*. AAAI Press, pp. 53–60.

Allen, J., G. Ferguson, and A. Stent (2001). An architecture for more realistic conversational systems. *Proceedings of the Conference on Intelligent User Interfaces (IUI-01)*, Santa Fe, NM, January 14-17, pages 1–8.

Burstein, M., G. Ferguson, and J. Allen (2000). *Integrating Agent-Based Mixed-Initiative Control with an Existing Multi-Agent Planning System*. Technical Report 729, Computer Science Dept., University of Rochester, May.

Ferguson, G., J. Allen, and B. Miller (1996). TRAINS-95: Towards a Mixed-Initiative Planning Assistant. *Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96)*. Edinburgh, Scotland, 29-31 May, pp. 70–77.

Ferguson, G. and J. Allen (1998). TRIPS: An Intelligent Integrated Problem-Solving Assistant. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. Madison, WI, 26-30 July, pp. 567–573.

Stent, A. (2001). *Dialogue Systems as Conversational Partners: Applying Conversation Acts Theory to Natural Language Generation for Task-Oriented Mixed-Initiative Spoken Dialogue.* Ph. D. thesis, Dept. of Computer Science, University of Rochester.

5. Trips and Demonstrations

January, 1999	James Allen and George Ferguson attended the CoABS PI meeting in Las Vegas, NV.
June, 1999	George Ferguson attended the CoABS PI meeting in Northampton, MA, and demonstrated the TRIPS-CAMPS system.
September, 1999	James Allen and George Ferguson attended the ARPI PI meeting in Washington, DC, and demonstrated the TRIPS-Pacifica and TRIPS-CAMPS systems.
October, 1999	James Allen and George Ferguson attended the CoABS Science Fair in Washington, DC, and demonstrated the TRIPS-CAMPS system.
December, 1999	George Ferguson demonstrated the TRIPS-CAMPS system at the AFRL Scientific Advisory Board meeting in Rome, NY.
February, 2000	George Ferguson and James Allen attended the CoABS PI meeting in Atlanta.
June, 2000	George Ferguson visited BBN in Boston, MA.
July, 2000	George Ferguson attended AAAI-2000 in Austin, TX.
July-August, 2000	James Allen and George Ferguson attended the CoABS PI meeting in Boston, MA, and demonstrated the TRIPS-MIATA system.
January, 2001	James Allen, George Ferguson, Amanda Stent, and Nate Blaylock attended IUI-2001 in Santa Fe, NM, and presented both the keynote address and a paper on the collaborative agent architecture.
January-February, 2001	James Allen and George Ferguson attended the CoABS PI meeting in Miami, FL, and demonstrated the TRIPS-MIATA system.
July, 2001	James Allen attended the CoABS PI meeting in Nashua, NH.
September, 2001	George Ferguson presented a briefing at the CoABS PM transition meeting in Washington, DC.
January, 2002	James Allen and George Ferguson attended the CoABS PI meeting in Washington, DC.
February, 2002	James Allen and George Ferguson attended the DAML PI meeting in St. Petersburg, FL.
May, 2002	James Allen gave an invited talk at the FLAIRS conference in Penscola, FL, that described much of the work we have performed in the CoABS program.
July, 2002	James Allen attended the AAMAS conference in Bologna, Italy, and presented a paper on the collaborative problem-solving model.
October, 2002	George Ferguson and James Allen attended the DAML PI meeting in Portland, OR.

October, 2002 George Ferguson and James Allen attended the Third Intl. NASA Workshop on Planning and Scheduling for Space in Houston, TX, and presented a paper on collaborative agents for planning and scheduling.

Appendix A: Integrating Agent-Based Mixed-Initiative Control with an Existing Multi-Agent Planning System

Burstein, M., G. Ferguson, and J. Allen (2000). *Integrating Agent-Based Mixed-Initiative Control with an Existing Multi-Agent Planning System*. Technical Report 729, Computer Science Dept., University of Rochester, May.

Integrating Agent-Based Mixed-Initiative Control With An Existing Multi-Agent Planning System

Mark Burstein
BBN Technologies
burstein@bbn.com

George Ferguson and James Allen
University of Rochester
{ferguson,james}@cs.rochester.edu

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 729

March 2003

This work is supported by The Defense Advanced Research Projects Agency (DARPA) under contracts F30602-98-C-0168 and F30602-98-2-0133.

Abstract

One of the less appreciated obstacles to scaling multi-agent systems is understanding the impact of the role(s) that people will play in those systems. As we try to adapt existing software tools and agent-based applications to play supportive roles in larger multi-agent systems, we must develop strategies for coordinating not only the problem solving behavior of these agent communities, but also their information sharing and interactive behavior. Our research interest is in mixed-initiative control of intelligent systems [Burstein and McDermott, 1996; Burstein *et al.*, 1998; Ferguson *et al.*, 1996a] and, in particular, of interactive planning systems comprised of a heterogeneous collection of software agents. In this paper, we describe our experience constructing a prototype tool combining elements of TRIPS [Ferguson and Allen, 1998], an interactive, mixed-initiative agent-based planning architecture using spoken natural language dialogue, with the CAMPS Mission Planner, an interactive airlift scheduling tool developed for the Air Force [Emerson and Burstein, 1999], together with some related resource management agents representing other parts of the airlift planning organization. The latter scheduling tools were not originally designed to participate as part of a mixed-initiative, interactive agent community, but rather were designed for direct user interaction through their own GUIs. We describe some requirements revealed by this effort for effective mixed-initiative interaction in such an environment, including the role of explanation, the need for contextual information sharing among the agents, and our approach to intelligent invocation and integration of available agent capabilities.

1 Introduction

One of the less appreciated obstacles to scaling multi-agent systems is understanding how people will play roles in those systems, and the nature of their interactions with them. We expect that many multi-agent systems will evolve out of existing software tools, suitably "agentized" so that they have greater access to other software agent resources, and to other classes of users. To develop these kinds of agent-based applications, playing supportive roles in larger multi-agent systems, we must develop strategies for coordinating not only the problem solving behavior of these agent communities, but also their information sharing and interactive behavior with users.

Our research is on mixed-initiative (user/agent) control of intelligent systems [Burstein and McDermott, 1996; Burstein *et al.*, 1998; Ferguson *et al.*, 1996a], and, in particular, of interactive planning systems comprised of a heterogeneous collection of software agents. Over the past year, we have been experimenting in the space of mixed-initiative, multi-agent planning systems. We have developed a prototype mixed-initiative planning tool for airlift scheduling by integrating elements of TRIPS, The Rochester Interactive Planning System, an agent-based, interactive, mixed-initiative planning system using spoken natural language dialogue [Ferguson and Allen, 1998], with the CAMPS Mission Planner, an interactive airlift scheduling tool developed for the Air Force [Emerson and Burstein, 1999], together with some related resource management agents representing other parts of the airlift planning organization.

This effort had a number of goals. First, we wished to demonstrate the relative ease with which the TRIPS agent architecture could be adapted to a new planning domain, and to interaction with a new back-end planner. Second, we sought to understand what would be required for an effective agent ACL interface to a scheduling or planning tool that had its own GUI and was not developed for multi-modal mixed-initiative interaction. Third, we wished to develop a model for the problem solving agent that could mediate between the user and a set of "back office" planning agents. The last of these goals is an ongoing activity.

2 Background

Our work involves the integration of three existing systems, each of which is described briefly in this section.

TRIPS, The Rochester Interactive Planner System [Ferguson and Allen, 1998] is a multi-agent system that includes agents for interacting with the user in spoken natural language (*e.g.*, speech recognition, parsing, reference resolution, discourse management, speech generation, *etc.*) as well as knowledge-based agents

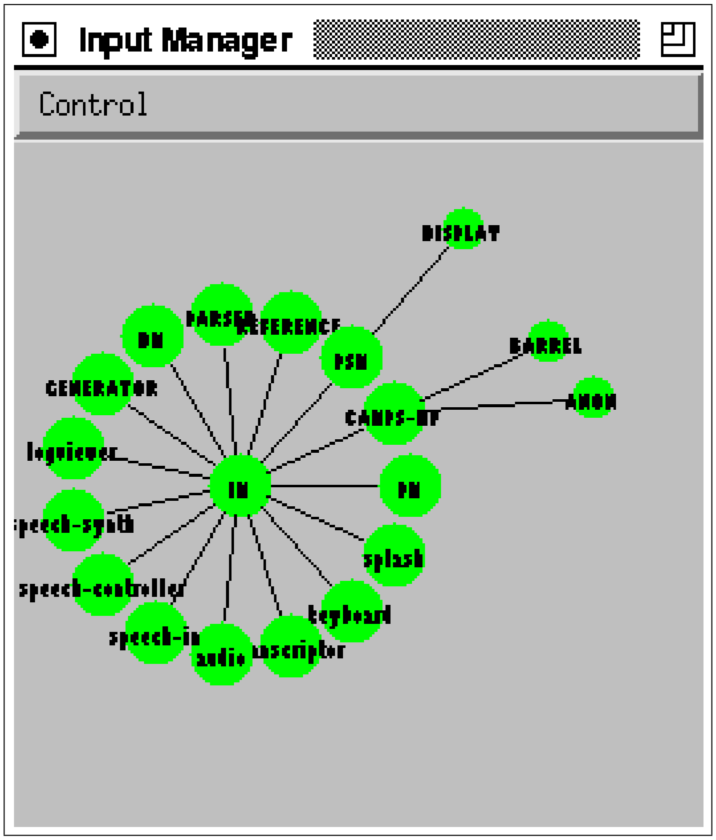


Figure 1: TRIPS Facilitator display

that provide task and domain expertise to assist the user (*e.g.*, planning, plan recognition, route-finding, scheduling, *etc.*). It is the latest in a series of mixed-initiative systems produced by an extended multi-year research program at the University of Rochester that began with the TRAINS system [Allen *et al.*, 1995; Ferguson *et al.*, 1996b]. The agents collectively provide a multi-modal interface by which users can discuss and develop plans through mixed-initiative interaction with what appears to them to be a single intelligent agent. The agents interact to produce this behaviour by exchanging messages using the KQML agent communication language [Finin *et al.*, 1993; Labrou and Finin, 1997], operating in a hub-based architecture provided by the TRIPS Facilitator. Figure 1 shows the Facilitator's GUI view as it appears in the integrated TRIPS/CAMPS system discussed in this paper (secondary nodes indicate a proxying relationship between agents). Figure 2 shows the logical organization of the agents involved in the TRIPS/CAMPS system.

Previously, TRIPS/TRAINS has been demonstrated for planning in domains such as non-combatant evacuations (NEOs), where a user and the system would

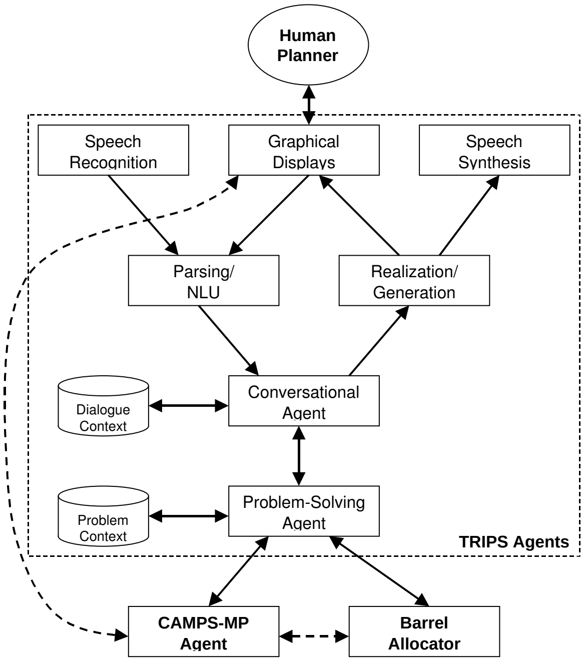


Figure 2: TRIPS/CAMPS architecture (logical organization)

jointly develop a plan to evacuate people from a fictitious island (Pacifica) in the face of an approaching hurricane. The user's task was to plan the evacuation using a variety of vehicles with varying capabilities, under constraints such as time, cost, weather, *etc.* The modality management agents (speech and NL processing, gesture capture) translate user actions into a set of possible *communicative acts* that are interpreted by the *Conversational Agent*. This agent interacts with the *Problem-Solving Agent* to find a plausible interpretation of what the user has just said and/or done. They then jointly determine an appropriate response by coordinating domain-level reasoners such as planners and schedulers. The Problem Solving Agent manages the invocation of the back-end agents and coordinates their responses.

TRIPS supports a wide range of speech acts, ranging from direct requests (*e.g.*, "show me the map"), questions ("Where are the transports?"), suggestions ("Let's use a helicopter instead."), acceptances, rejections, and a range of social acts. The Conversational Agent is domain independent. It is driven by a set of rules that identify possible interpretations intended by a user, and plans an appropriate kind of system response for each. Each interpretation/response is ranked and a single response is selected. The Problem Solving agent (PS) is responsible for maintaining the problem solving context, interacting with the Conversational Agent to decide which interpretations of user actions are plausible, and plan for the execution of the selected action by communicating with the appropriate back end agent(s).

The specialized back-end agents in TRIPS-98 and earlier incarnations included

U: Show me a map of pacifica.
T: Ok. *(shows the map)*
U: Where are the people?
T: There are two groups of people at Exodus, two at Calypso, . . .
U: Use a truck to get the people from Calypso to Delta.
T: Ok. *(show timeline view of plan under development in Plan Viewer)*
U: How long will that take?
T: It will take four hours and fifty minutes.
U: What if we went along the coast instead?
T: That option will take six hours and thirty-seven minutes under normal conditions.
U: Forget it.
. . .

Figure 3: Sample Dialogue with TRIPS-98 (excerpt)

an incremental (repair-oriented) temporal planner, a route finder, a scheduler, a simulator that represented the changing world state, and a temporal knowledge base agent. These various agents were invoked by the problem solving agent in response to user inputs. A short excerpt from a typical session is shown in Figure 3.

The CAMPS Mission Planner [Emerson and Burstein, 1999] is one of several prototype scheduling tools developed jointly by BBN Technologies, Kestrel Institute and CMU for the US Air Force. It is currently undergoing refinement and integration in preparation for deployment in about a year's time. As part of the DARPA Control of Agent-Based Systems Program, BBN has "agent-ized" the Mission Planner (MP) for experimental use in mixed-initiative multi-agent systems. The MP takes as inputs a set of "requirements," each consisting of quantities of cargo and people to be moved from one airport to another during some time interval. It produces detailed schedules specifying the times at which an aircraft of some type will fly from where they are based to pick up this cargo and carry it to its destination, and then return to home base. Requirement sets can be quite large, numbering hundreds or even thousands of elements, and hundreds of tons of cargo, from tens of locations. Numerous constraints must be satisfied simultaneously to produce valid schedules, which the scheduler can do in seconds or minutes.

A simplistic view of the task faced by a user of the Mission Planner is, given a set of requirements, is to specify a set of suitable aircraft resources, the ports to be made available for refueling (or locations for aerial refueling), should that be necessary, and to ensure that the schedule produced moves all of the requirements by their due dates. The scheduler will fail to schedule flights for all of the cargo requirements if there are constraint violations, or insufficient resources provided, in which case some cargo may be scheduled to arrive late or not at all. As originally developed, the CAMPS Mission Planner had a traditional graphical user interface for specifying input parameters, and a variety of views of the product schedules produced, including maps, tables and GANTT charts. All user interactions were by keyboard and mouse.

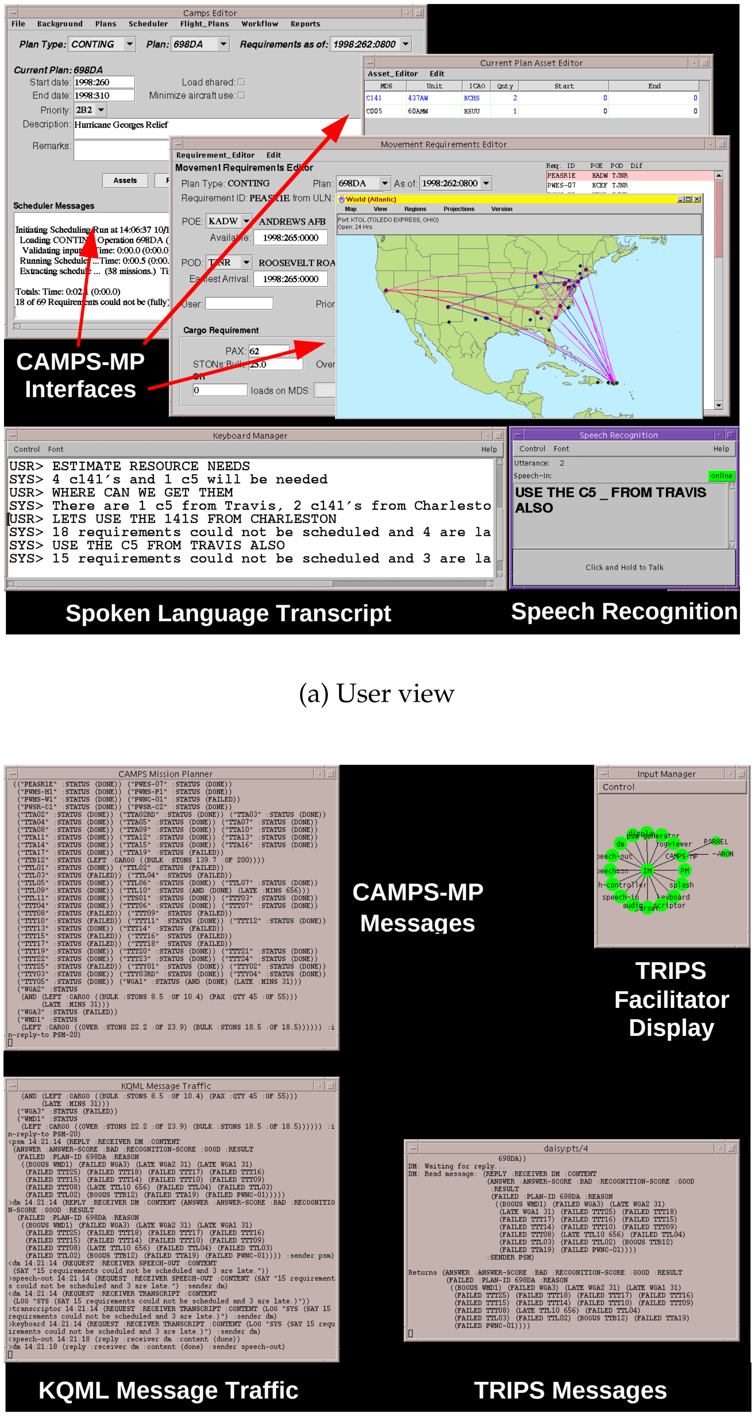
A second scheduling tool, called the **Barrel Allocator** [Becker and Smith, 2000] was developed jointly by CMU and BBN for another community of users (called Barrelmasters for historical reasons) within the same Air Force planning organization. It is used to manage all of the cargo aircraft available from different bases around the country. This tool takes as input the schedules produced by planners once those plans have been committed to, and finds an allocation of aircraft to missions that makes the most productive use of the limited available resources. As more airlift missions are planned by different planners than there are aircraft to fly them all, and because they are often planning far in advance, planners often plan against notional resources, rather than checking for where aircraft will be available. This tool is used to commit particular bases to particular missions, by priority, proximity, and availability, and if necessary reschedule flights to use aircraft from different bases then originally planned. To represent the interaction between the planners and the allocators, a simplified version of the Barrel system was developed into an agent for use in the prototype system.

3 Preliminary Integration Experiment

In developing this demonstration system, our approach was to provide a straightforward API to the MP and BARREL agents using the KQML agent communications language, and to collect and catalog the reasoning and information interchange issues of the various agents forming the complete system. This included problems arising in interpretation and reformulation of user intent, and the planning of requests to the various back-end, airlift-domain-specific agents.

Figure 4 shows some of the graphical views as seen from a user's perspective, as well as a "behind the curtain" view of the system's activity and inter-agent message traffic. Figure 5 shows a sample dialogue with the system. All of the dialogue was verbal, and the conversation was managed by the TRIPS agents. The graphic displays were generated by the CAMPS-MP agent, using pre-existing views, with one exception. While a user could have accomplished much the same result by interaction with the CAMPS-MP GUIs, many steps would have required tedious entry of information through dialog windows.

The sample dialogue illustrates much of the capability of the combined system at present. Using spoken language dialogue, users can quickly arrive at a solution to typical scheduling problems, in many ways more easily and more naturally than through the GUI. They can select problems to work on, request advice on the



Plan Type: CONTING   Plan: 698DA   Requirements as of: 1998:262:0800

Current Plan: 698DA
Start date: 1998:260   Load shared: □
End date: 1998:310   Minimize aircraft use: □
Priority: 2B2
Description: Hurricane Georges Relief
Remarks:

Scheduler Messages

```
Initiating Scheduling Run at 14:06:37 10/1
 Loading CONTIN  Operation 698DA (
  Validating input  Time: 0:00.0 (0:00.0)
 Running Schedule ...Time: 0:00.5 (0:00.
 Extracting schedule ...  (38 missions.)  Ti

Totals: Time: 0:02.1 (0:00.0)
18 of 69 Requirements could not be (fully)
```

MDS	Unit	ICAO	Qnty	Start	End
C141	437AW	KCHS	2	0	0
C005	60AMW	KSUU	1	0	0

Movement Requirements Editor
Plan Type: CONTING   Plan: 698DA   As of: 1998:262:0800
Requirement ID: PEASR1E from ULN:
POE: KADW   ANDREWS AFB
Available: 1998:265:0000
POD: TJNR   ROOSEVELT ROA
Earliest Arrival: 1998:265:0000
User: Prior

Cargo Requirement
PAX: 62
STONs:Bulk: 25.0   Over
0   loads on MDS:

Req. ID	POE	POD	Dif
PEASR1E	KADW	TJNR	
PWES-07	KCEF	TJNR	

Port: KTOL (TOLEDO EXPRESS, OHIO)
Open: 24 Hrs.

```
USR> ESTIMATE RESOURCE NEEDS
SYS> 4 c141's and 1 c5 will be needed
USR> WHERE CAN WE GET THEM
SYS> There are 1 c5 from Travis, 2 c141's from Charlesto
USR> LETS USE THE 141S FROM CHARLESTON
SYS> 18 requirements could not be scheduled and 4 are la
SYS> USE THE C5 FROM TRAVIS ALSO
SYS> 15 requirements could not be scheduled and 3 are la
```

Utterance: 2
Speech–In:
USE THE C5 _ FROM TRAVIS ALSO

(a) User view

```
(("PEASR1E" :STATUS (DONE)) ("PWES-07" :STATUS (DONE))
 ("PWMS-H1" :STATUS (DONE)) ("PWMS-P1" :STATUS (DONE))
 ("PWMS-W1" :STATUS (DONE)) ("PWNC-01" :STATUS (FAILED))
 ("PWSR-C1" :STATUS (DONE)) ("PWSR-C2" :STATUS (DONE))
 ("TTA02" :STATUS (DONE)) ("TTA02RD" :STATUS (DONE)) ("TTA03" :STATUS (DONE))
 ("TTA04" :STATUS (DONE)) ("TTA05" :STATUS (DONE)) ("TTA07" :STATUS (DONE))
 ("TTA08" :STATUS (DONE)) ("TTA09" :STATUS (DONE)) ("TTA10" :STATUS (DONE))
 ("TTA11" :STATUS (DONE)) ("TTA12" :STATUS (DONE)) ("TTA13" :STATUS (DONE))
 ("TTA14" :STATUS (DONE)) ("TTA15" :STATUS (DONE)) ("TTA16" :STATUS (DONE))
 ("TTA17" :STATUS (DONE)) ("TTA19" :STATUS (FAILED))
 ("TTB12" :STATUS (LEFT :CARGO ((BULK :STONS 139.7 :OF 200))))
 ("TTL01" :STATUS (DONE)) ("TTL02" :STATUS (FAILED))
 ("TTL03" :STATUS (FAILED)) ("TTL04" :STATUS (FAILED))
 ("TTL05" :STATUS (DONE)) ("TTL06" :STATUS (DONE)) ("TTL07" :STATUS (DONE))
 ("TTL09" :STATUS (DONE)) ("TTL10" :STATUS (AND (DONE) (LATE :MINS 656)))
 ("TTL11" :STATUS (DONE)) ("TTS01" :STATUS (DONE)) ("TTT03" :STATUS (DONE))
 ("TTT04" :STATUS (DONE)) ("TTT06" :STATUS (DONE)) ("TTT07" :STATUS (DONE))
 ("TTT08" :STATUS (FAILED)) ("TTT09" :STATUS (FAILED))
 ("TTT10" :STATUS (FAILED)) ("TTT11" :STATUS (DONE)) ("TTT12" :STATUS (DONE))
 ("TTT13" :STATUS (DONE)) ("TTT14" :STATUS (FAILED))
 ("TTT15" :STATUS (FAILED)) ("TTT16" :STATUS (FAILED))
 ("TTT17" :STATUS (FAILED)) ("TTT18" :STATUS (FAILED))
 ("TTT19" :STATUS (DONE)) ("TTT20" :STATUS (DONE)) ("TTT21" :STATUS (DONE))
 ("TTT22" :STATUS (DONE)) ("TTT23" :STATUS (DONE)) ("TTT24" :STATUS (DONE))
 ("TTT25" :STATUS (FAILED)) ("TTY01" :STATUS (DONE)) ("TTY02" :STATUS (DONE))
 ("TTY03" :STATUS (DONE)) ("TTY03RD" :STATUS (DONE)) ("TTY04" :STATUS (DONE))
 ("TTY05" :STATUS (DONE)) ("WGA1" :STATUS (AND (DONE) (LATE :MINS 31)))
 ("WGA2" :STATUS
  (AND (LEFT :CARGO ((BULK :STONS 8.5 :OF 10.4) (PAX :QTY 45 :OF 55)))
       (LATE :MINS 31)))
 ("WGA3" :STATUS (FAILED))
 ("WMD1" :STATUS
  (LEFT :CARGO ((OVER :STONS 22.2 :OF 23.9) (BULK :STONS 18.5 :OF 18.5)))))) :i
n-reply-to PSM-20)
```

```
  (AND (LEFT :CARGO ((BULK :STONS 8.5 :OF 10.4) (PAX :QTY 45 :OF 55)))
       (LATE :MINS 31)))
 ("WGA3" :STATUS (FAILED))
 ("WMD1" :STATUS
  (LEFT :CARGO ((OVER :STONS 22.2 :OF 23.9) (BULK :STONS 18.5 :OF 18.5)))))) :i
n-reply-to PSM-20)
<psm 14:21:14 (REPLY :RECEIVER DM :CONTENT
 (ANSWER :ANSWER-SCORE :BAD :RECOGNITION-SCORE :GOOD :RESULT
  (FAILED :PLAN-ID 698DA :REASON
   ((BOGUS WMD1) (FAILED WGA3) (LATE WGA2 31) (LATE WGA1 31)
    (FAILED TTT25) (FAILED TTT18) (FAILED TTT17) (FAILED TTT16)
    (FAILED TTT15) (FAILED TTT14) (FAILED TTT10) (FAILED TTT09)
    (FAILED TTT08) (LATE TTL10 656) (FAILED TTL04) (FAILED TTL03)
    (FAILED TTL02) (BOGUS TTB12) (FAILED TTA19) (FAILED PWNC-01)))))
>dm 14:21:14 (REPLY :RECEIVER DM :CONTENT (ANSWER :ANSWER-SCORE :BAD :RECOGNITIO
N-SCORE :GOOD :RESULT
  (FAILED :PLAN-ID 698DA :REASON
   ((BOGUS WMD1) (FAILED WGA3) (LATE WGA2 31) (LATE WGA1 31)
    (FAILED TTT25) (FAILED TTT18) (FAILED TTT17) (FAILED TTT16)
    (FAILED TTT15) (FAILED TTT14) (FAILED TTT10) (FAILED TTT09)
    (FAILED TTT08) (LATE TTL10 656) (FAILED TTL04) (FAILED TTL03)
    (FAILED TTL02) (BOGUS TTB12) (FAILED TTA19) (FAILED PWNC-01)))) :sender psm)
<dm 14:21:14 (REQUEST :RECEIVER SPEECH-OUT :CONTENT
 (SAY "15 requirements could not be scheduled and 3 are late."))
>speech-out 14:21:14 (REQUEST :RECEIVER SPEECH-OUT :CONTENT (SAY "15 requirement
s could not be scheduled and 3 are late.") :sender dm)
<dm 14:21:14 (REQUEST :RECEIVER TRANSCRIPT :CONTENT
 (LOG "SYS (SAY 15 requirements could not be scheduled and 3 are late.)"))
>transcriptor 14:21:14 (REQUEST :RECEIVER TRANSCRIPT :CONTENT (LOG "SYS (SAY 15
requirements could not be scheduled and 3 are late.)") :sender dm)
>keyboard 14:21:14 (REQUEST :RECEIVER TRANSCRIPT :CONTENT (LOG "SYS (SAY 15 requ
irements could not be scheduled and 3 are late.)") :sender dm)
<speech-out 14:21:18 (reply :receiver dm :content (done))
>dm 14:21:18 (reply :receiver dm :content (done) :sender speech-out)
```

```
                     698DA))
DM: Waiting for reply...
DM: Read message: (REPLY :RECEIVER DM :CONTENT
                   (ANSWER :ANSWER-SCORE :BAD :RECOGNITION-SCORE :GOOD
                    :RESULT
                    (FAILED :PLAN-ID 698DA :REASON
                     ((BOGUS WMD1) (FAILED WGA3) (LATE WGA2 31)
                      (LATE WGA1 31) (FAILED TTT25) (FAILED TTT18)
                      (FAILED TTT17) (FAILED TTT16) (FAILED TTT15)
                      (FAILED TTT14) (FAILED TTT10) (FAILED TTT09)
                      (FAILED TTT08) (LATE TTL10 656) (FAILED TTL04)
                      (FAILED TTL03) (FAILED TTL02) (BOGUS TTB12)
                      (FAILED TTA19) (FAILED PWNC-01))))
                   :SENDER PSM)

Returns (ANSWER :ANSWER-SCORE :BAD :RECOGNITION-SCORE :GOOD :RESULT
         (FAILED :PLAN-ID 698DA :REASON
          ((BOGUS WMD1) (FAILED WGA3) (LATE WGA2 31) (LATE WGA1 31)
           (FAILED TTT25) (FAILED TTT18) (FAILED TTT17) (FAILED TTT16)
           (FAILED TTT15) (FAILED TTT14) (FAILED TTT10) (FAILED TTT09)
           (FAILED TTT08) (LATE TTL10 656) (FAILED TTL04)
           (FAILED TTL03) (FAILED TTL02) (BOGUS TTB12) (FAILED TTA19)
           (FAILED PWNC-01))))
```

(b) System view

Figure 4: User (a) and system (b) views of TRIPS/CAMPS system

001 U: **`Hello.`**

002 T: Hey there.

003 U: **`Show me the active plans.`**

004 T: Ok.

[CAMPS-MP displays a list of plans under development and returns their IDs and descriptions to TRIPS.]

005 U: **`Let’s work on the Georges plan.`**

006 T: Alright.

[CAMPS-MP main viewer focusses on Hurricane Georges deployment plan.]

007 U: **`Estimate resource needs.`**

[TRIPS requests the estimate from CAMPS-MP.]

008 T: Four C141s and 1 C5 will be needed.

[TRIPS forms its utterance using CAMPS-MP’s answer.]

009 U: **`Where can we get them?`**

[TRIPS sends a set *of requests to the BARREL agent corresponding to the two sets of (nominal) aircraft identified as “them”.]*

010 T: There are one C5 from Travis, two C141’s from McGuire, and three C141’s from Charleston.

[TRIPS integrates the BARREL’s replies and formulates the response.]

011 U: **`Let’s use two 141s from McGuire.`**

[TRIPS interprets “use” as a request to provide the given aircraft as resources for scheduling, using the `AUGMENT-PLANSET` *request to CAMPS-MP.]*

012 T: Ok. You still need one C5 and two C141s before we can schedule.

[TRIPS uses the original resource estimate to prompt the user for additional information before invoking the scheduler.]

013 U: **`Use the C5 from Travis also.`**

014 T: Ok. You still need two C141s before we can schedule.

015 U: **`And use the 141s from Charleston.`**

[Once TRIPS believes the scheduling problem is sufficiently specified, it requests that CAMPS-MP invoke the scheduler.]

016 T: 12 requirements could not be scheduled.

[CAMPS-MP indicates the status of the plan after scheduling, which TRIPS summarizes for the user.]

017 U: **`Show them to me.`**

018 T: Ok.

[CAMPS-MP displays the failed requirements in a table view.]

019 U: **`Explain PWNC-1.`**

[“PWNC-1” is the ID of one requirement to move cargo from Washington County Regional Airport to Puerto Rico.]

020 T: C5s can’t land at Washington County Regional.

Figure 5: Annotated TRIPS/CAMPS sample dialogue

021 U: **Let’s use a C17 instead.**
[This aircraft type can use a shorter runway.]

022 T: One requirement is still unscheduled.
[TRIPS locates a C17 by querying the BARREL, provides it to CAMPS-MP as a resource, then invokes the scheduler again.]

023 U: **Explain it.**
[TRIPS identifies “it” as the failed requirement.]

024 T: C17s, C5s, and C141s cannot fly non-stop from Elmendorf to Roosevelt Roads.
[Elmendorf AFB (source of the requirement) is in Alaska, Roosevelt Roads (destination of the requirement) is in Puerto Rico. The distance between them is too far for the planes to fly without refueling.]

025 U: **Show me a map.**

026 T: Ok.
[TRIPS invokes the CAMPS-MP map viewer with all the routes displayed.]

027 U: **Use Charleston as an en-route stop.**
[TRIPS makes Charleston available to CAMPS-MP as a refueling resource, and reinvokes the scheduler.]

028 T: Ok. All requirements are scheduled.

Figure 5: Annotated TRIPS/CAMPS sample dialogue (cont.)

resources required to solve the problem, suggest particular assets to use, request views of the state of problem definition and of the solutions produced, and receive explanations when things don’t go as expected. TRIPS manages the interaction with CAMPS-MP and the BARREL agent, and automatically invokes the scheduler when it is appropriate.

This first-cut integration of TRIPS and CAMPS agents was developed in a relatively short period of time (approximately three months) in part because of the modular, agent-oriented construction of TRIPS itself. Aside from small efforts for extending the vocabulary and language understanding modules, the main effort was in “agent-ifying” the aiflift domain agents, CAMPS-MP and BARREL, and in developing a specialized Problem Solving Agent that could do the required interpretation and translation of user actions and airlift agent responses, in the context of the airlift mission planner’s task.

The ongoing scientific focus of our effort is now on understanding the capabilities needed in a problem solver that can manage this kind of interaction. In particular, we are now developing a model of the overall problem solving task, that can be used by a more general purpose problem solving agent to interpret user requests in terms of a more declaratively described domain model and relate these to models of the back-end agents.

4 Major Technical Issues

The main focus of this paper is on "lessons learned" about the level of information sharing required of the domain specific agents, and the kinds of reformulation and translation of user requests that is required to facilitate naturalistic, mixed-initiative dialogues with fieldable domain-specific planning/scheduling tools like that shown in Figure 4. Although there are a few other systems, such as Quickset [Cohen *et al.*, 1997] and TRIPS itself, that have demonstrated some of these capabilities, what is new here is the coupling of these largely domain-independent interaction modality-management agents with domain-specific planning systems that were not originally designed for such coordinated, mixed-initiative, multi-agent behavior.

4.1 "Agent-izing" the Mission Planner

The CAMPS Mission Planner system is itself composed of a GUI written in JAVA and a server written in LISP. It was turned into a single agent by providing the LISP server with a KQML interface. We developed a KQML API that provided essentially a programmatic means of invoking the behaviors available through the scheduler's graphical user interface. This was done without explicit reference to the internal representations manipulated by and shared among the TRIPS agents. Table 1 shows a summary of the message signatures handled by the CAMPS-MP agent. It includes a set of commands for manipulating the various views provided by the CAMPS-MP interface, a set of commands for describing new problems and revising old problems to be posed to the scheduler, a request for invoking the scheduler, and queries for analyses of scheduler results and explanations of problems found in those results.

The main information passing messages (`NEW-PLANSET`, `REVISE-PLANSET`, and `AUGMENT-PLANSET`) all take as keyword arguments all of the different kinds of data elements that define problems for the Mission Planner to solve. `NEW-PLANSET` is used to create a new problem from scratch, while `AUGMENT-PLANSET` is used to add ports, resources or requirements to an existing problem. `REVISE-PLANSET` is used to replace one of the previously specified sets of values. We believe that this interface is representative of the kind of interface that one will get in practice when "wrapping" a legacy system. As such we saw it as a reasonable target for the problem solver agent that it was to interact with.

We did add functionality to the CAMPS-MP system in several places to facilitate the kind of dialogue illustrated in Figure 5. The most important of these was that we developed a capability to estimate the resource levels needed to solve the scheduling problem. This capability is now being considered as a requirement for the CAMPS Mission Planner that will be deployed next year (without the natural

KQML Performative	Content Form
REQUEST	(OPEN-VIEWER :VIEWER <VIEWER>)
REQUEST	(FOCUS-VIEWER :VIEWER <VIEWER> ...)
REQUEST	(CLOSE-VIEWER :VIEWER <VIEWER>)
REQUEST	(DISPLAY-TABLE :TITLE <STRING> :QUERY <SQL>)
REQUEST	(MAP-FOCUS :LATITUDE <LAT> :LONGITUDE <LON> :ZOOM <FLOAT> :PLAN-ID <ID>)
INFORM	(NEW-PLANSET :PLAN-ID <ID> :START-DATE <DATE> :END-DATE <DATE> :PRIORITY <PRI> :REQUIREMENTS (<REQUIREMENT>*) :AVAILABLE-ASSETS ((<ACTYPE> <QTY> <UNITID> <START> <END>)*) :AVAILABLE-PORTS ((<LOCID> <NAME> <ENROUTE?> ...)*) ...)
INFORM	(REVISE-PLANSET *keywords same as* NEW-PLANSET)
INFORM	(AUGMENT-PLANSET *keywords same as* NEW-PLANSET)
ASK-ONE	(FLIGHT-PLAN :PLAN-ID <ID>)
ASK-ALL	(LIST-PLANSETS :FIELDS ...)
REQUEST	(SHOW-PLANSETS :PLAN-TYPE <str> :NEW-SINCE <time> :CHANGED-SINCE <time>...)
REQUEST	(ANALYZE-PLAN :PLAN-ID <ID>)
REQUEST	(SHOW-UNSATISFIED-REQUIREMENTS :PLAN-ID <ID>)
REQUEST	(SCHEDULE-AIRLIFT :PLAN-ID <ID>)
REQUEST	(ESTIMATE-RESOURCES :PLAN-ID <ID>)
REQUEST	(EXPLAIN :OBJECT (UNSATISFIED :REQ-ID <ID>))

Table 1: Summary of KQML interface to CAMPS-MP

language interface). We also added several new graphical views to summarize in tabular form queries that seemed natural to ask for in English, such as the view showing the "open problems list" that is requested at the beginning of the sample dialogue.

4.2 Supporting Mixed-Initiative Explanation

Unlike the legacy scheduling system that the CAMPS Mission Planner is being developed to replace, CAMPS-MP included a capability to generate simple explanations of scheduler failures prior to this integration experiment. This was important for the mixed-initiative interactions we sought to demonstrate. CAMPS-MP generates explanations for unscheduled or late requirements by looking for necessary constraints in the problem statement (the scheduler inputs) that make it impossible for the scheduler to successfully schedule flights for all of the requirements. These constraints are those that must be satisfied to form a sequences of flights by an aircraft going from its home base to the cargo to be moved, loading and move the cargo to its destination, and then returning the aircraft to its base. The constraints currently checked for explanation generation are:

- The availability of an aircraft with appropriate carrying capacity for the kind of cargo to be moved. The natural user response to this kind of constraint failure is to introduce a new kind of asset as an available resource in the statement of the problem.

- The capability of an appropriate aircraft to land at all ports along the route. This is checked by comparing the maximum runway length of the port to the minimum runway length required. Again, the appropriate repair is to introduce an aircraft of a type that can land at all ports along the route.

- The capability of the aircraft to fly the distances between each port along its route without refueling. The CAMPS-MP scheduler will ordinarily attempt to land the aircraft at an en-route stop, or use an aerial refueling if either of those options is provided at a location along the route. The repair here is to introduce or designate such a location as part of the scheduler's problem description.

Of course, there are a number of other constraints that can interact to produce scheduling failures (see Emerson and Burstein, 1999). Explaining these more complex failures is an ongoing topic of the CAMPS-MP development effort. Given that this preliminary explanation facility already existed, the main change required for the integration of TRIPS and CAMPS-MP was the production of those explanations in a declarative representation, suitable for inclusion in a KQML message, rather than directly as “canned” text messages to the user. For the limited set of constraints now handled, this was easily accomplished. Responses to KQML queries of the form

```
(ASK-ALL
  :CONTENT (EXPLAIN :OBJECT (UNSCHEDULED :REQ-ID <ID>))
```

refer to these constraint failures by one of the following:

```
(NOT (CAN-LAND-AT <ac types> <port>)) |
(NOT (CAN-FLY-NONSTOP <ac types> <port1> <port2>)) |
(NOT (CAN-CARRY-CARGO <ac types> <reqid>))
```

and these are in turn translated into English by the TRIPS generator, where their content becomes part of the dialogue context. Ultimately, we would expect that the problem solver could make more direct use of this information, to suggest or even carry out the appropriate repairs, if no further user input is needed to proceed.

4.3 Context Sharing to Support Mixed-Initiative Interaction

Recall that our goal is to provide the user with a seamless view of the agents at their disposal, by making the system itself an intelligent agent that interacts with the user. A crucial issue in supporting this appearance while integrating legacy systems is the need to share contextual information among the agents. Logically, anything that the user sees becomes part of the shared dialogue context, and he or she can refer to it in subsequent communications with the system. In the TRIPS/CAMPS system, this issue arose during the design of the agentized Mission Planner’s API because the back-end domain agents hold much of the detailed knowledge of the problem state *and are responsible for displaying that information graphically to the user*. If the system had been engineered from scratch as a single integrated system, so that all user displays were handled by a distinct agent, the problem would have still occurred, but the required communications patterns would have been different.

Consider what happens when the user requests a display of information, such as the list of the active plans needing to be worked on, or a map of the scheduled routes. All of the information displayed in the table generated by CAMPS-MP becomes subject to reference as part of the current dialogue context. The model of the problem solving task we are developing would suggest that the user actions following a request for the list of active plans is either to select one of them to work on, or to ask for more information about one or several of them. In either case, the problem solver, the reference resolution agent, and perhaps even the speech and language agents require that the list of plans, including their IDs and names, are known and are now salient.

To manage this in our prototype, we used the design principle that all messages requesting that displays be shown return, in addition to an acknowledgment, a declaratively represented summary of the displayed information. Alternatively, one might require that all agents displaying information be capable of being queried for the content of their displays.

The larger issue is how this kind of information is uniformly made available to all of the agents requiring it. We know of no general solution to this. In agent architectures like the star topologies of TRIPS and OAA [Cohen *et al.*, 1994], this information could, in principle, be broadcast to all agents who care to use it. In network communications architectures like RETSINA [Sycara *et al.*, 1996], one might require that some agent or set of agents serve as a blackboard or clearinghouse for the information. Beyond the question of who gets the information, a key remaining problem is that the descriptions of contextual items must include semantic information. For example, the TRIPS components of our system need to know not only that "PWNC-01" is being displayed, but also that it is a transport requirement, that its status is unscheduled, and so on.

4.4 Problem Solving: Plan Recognition, Task Allocation, and Agent Management

Most of the burden involved in actually helping the user with their task falls to the Problem Solving (PS) agent. Its responsibilities in the TRIPS/CAMPS architecture can be divided into three levels.

First, it is responsible for interacting with the Conversational Agent when the latter determines that the interpretation of the current utterance involves domain-level problem-solving. At this level, the Problem Solver checks whether the proposed interpretation is plausible in the current problem-solving context. In general, this requires some form of plan recognition. Although for the simplified airlift mission planner's task used in our integration experiment, simple heuristics were sufficient, we are now building a more complete model of the task suitable for use in this interpretation phase.

Second, if the interpretation seems to make sense, the Problem Solver must determine how to meet its obligation to respond to it. This might involve modifying the problem solving state, as in a suggestion to extend or modify the parameters of the problem (*e.g.*, "Use the C5 from Charleston also"), or determining the answer to a query (*e.g.*, "Where can we get them [the assets]?"), and so on. At this level, the Problem Solver needs to use its knowledge of the problem solving state and of the capabilities of the agents at its disposal to produce a plan for using those agents to fulfill its obligations. In most cases, this requires matching agent capabilities against outstanding obligations, using some general principles for decomposing tasks into pieces those agents can handle.

An interesting special case here is interactive plan repair. When the result of the previous actions by the user and/or the system is the discovery of a problem with the partial plan produced, the user may ask for (or, better yet, the system may generate on its own) an explanation of the failure. The explanation sets the context for some particular class(es) of repairs, as described in Section 4.2 above. In the sample dialogue with the present system, essentially context-independent interpretation of the user response "Use a C17 instead" allows the planning to proceed, but a seemingly subtle change to "Use a C17" would be ambiguous, and the system could not proceed. Without a model of the plausible repairs for the failure, the system would not be able to prefer the substitution interpretation over the addition of additional assets to the existing specification.

Finally, at the third level, the Problem Solver must execute this plan and actually interact with the agents to obtain the information needed to fulfill its obligations. This involves translating between the general-purpose representation used by the Problem Solver and whatever API is available from the agent being invoked. Typically, multiple messages must be sent, since the Problem Solver's obligations come from the expressive language interface, whereas individual agent's APIs are usually quite simple. An example here is the interaction with the BARREL agent to find available assets ("Where can we get them?"). The Problem Solver, having resolved the reference to the list of notional assets estimated by CAMPS-MP to be needed, must generate a sequence of queries to the BARREL to determine where each type of asset can be found. The replies must be gathered and interpreted, and the Problem Solver moves on to the next step in its plan (barring an unexpected response or error indication, which would abort the plan or trigger other remedial action, such as entering into a sub-dialogue with the user).

Once all the necessary information has been obtained, a suitable response must be produced for use by the Conversational Agent in maintaining the dialogue context and by the realization components for presentation to the user. The Problem Solver returns both the plausibility estimate for each interpretation of the user's intent, as determined in Step 1, and an answer score indicating the quality of the solution found in Steps 2 and 3. The Conversational Agent uses both of these values to select the preferred interpretation of the user's utterance and produce an appropriate response. When the response requires use of another agent's display capabilities (as with CAMPS displaying a map or table), the Problem Solver manages the invocation of the other agent(s), with suitable translation and acknowledgments.

In the initial demonstration system, Steps 2 and 3 of this process are not performed as generally as described above. Instead, the Problem Solver has a single model of the airlift mission planner's task together with how the CAMPS-MP and BARREL agents fit into that task. We are now developing a framework to support declarative representation of the user's problem solving task and of the capabilities of the back-end agents, thereby permitting a more direct and general implementation of the problem solving model described above. One of the results of this work will be a system that can more easily be used in a variety of mixed-initiative tasks and with a dynamically changing set of back-end agents.

5 Conclusions and Directions

We have described an initial integration of the TRIPS agent framework for multi-modal mixed-initiative control of planning support agents with several domain-specific agents representing aspects of the airlift planning domain that were not originally designed to work in such a framework. Our larger objectives here are to develop the framework to support appropriate utilization of a wider variety of agents in support of user objectives, and to utilize those agents more proactively when the opportunities arise.

In the process, we have uncovered, or, in some cases, rediscovered, requirements for both the problem solver mediating between the user and these domain agents, and for the capabilities of the domain agents that will be suitable for participating in such systems. In particular, for "legacy agents," we have discussed the importance of information sharing between agents that can present information to the user and the dialogue components, and the important role that a capability to generate explanations can play in furthering the cooperative problem solving of the user+agents team.

For the problem solver, we have again motivated the need for both capability as well as *use* models for the agents it is directing. We believe that these models will be utilized primarily in a reactive planning framework, with some outstanding issues. First, the translation of user requests into subtasks for other agents in general requires some amount of reformulation and decomposition to match the API of the target agents. In part, this is a generative planning task not unlike that found in systems for distributed information retrieval like SIMS and ARIADNE [Arens *et al.*, 1996; Ambite and Knoblock, 1997] It could perhaps be largely decoupled from the problem solver's role in context-sensitive intent interpretation and subgoal formation.

Second, and perhaps more problematic, is the development of a general framework for the Problem Solver's role in multi-modal response generation. Here, to be cooperative in a mixed-initiative sense, the problem solver must be capable of summarization (as when reporting the results of scheduling by characterizing the failures), and also of directing the user to view salient visualizations, when they are available through some agent at its disposal.

Finally, we see the development of task/use models in the problem solver as providing excellent opportunities for improving the initiative of such systems. At present, TRIPS/CAMPS takes initiative simply when it has enough information to attempt calling the scheduler (based on a description of the scheduling capability). A use model could predict when discovered failures should lead directly to an (unprompted) request of MP for an explanation, and even potentially to routine repairs, when no further discrimination is required of the user. These models could also perhaps be the locus for machine learning about appropriate times for initiative, if a system like TRIPS/CAMPS was routinely used to solve similar kinds of scheduling problems, as would undoubtedly be the case in the airlift domain.

References

[Allen *et al.*, 1995] James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum, "The TRAINS Project: A case study in defining a conversational planning agent," *Journal of Experimental and Theoretical AI*, 7:7–48, 1995, Also available as TRAINS Technical Note 93-4, Department of Computer Science, University of Rochester.

[Ambite and Knoblock, 1997] Jose-Luis Ambite and Craig A. Knoblock, "Planning by rewriting: Efficiently generating high-quality plans," In *Proceedings of the Fourteenth National Conference on Artifical Intelligence (AAAI-97)*, 1997.

[Arens *et al.*, 1996] Yigal Arens, Craig A. Knoblock, , and Wei-Min Shen, "Query reformulation for dynamic information integration," *Journal of Intelligent Information Systems*, 6(2/3):33–130, 1996.

[Becker and Smith, 2000] Marcel A. Becker and Stephen F. Smith, "Mixed-Initiative Resource Management: The AMC Barrel Allocator," In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, Breckenridge, CO, April 2000.

[Burstein *et al.*, 1998] Mark Burstein, Alice Mulvehill, and Steve Deutsch, "An Approach to Mixed-Initiative Management of Heterogeneous Software Agent Teams.," In *Proceedings of the Thirty-first Hawaiian International Conference on Systems Sciences (HICCS-98)*, Kona, HI, 6-9 January 1998.

[Burstein and McDermott, 1996] Mark H. Burstein and Drew McDermott, "Issues in the Development of Human-Computer Mixed-Initiative Planning," In B. Gorayska and J.L. Mey, editors, *Cognitive Technology*, pages 285–303. Elsevier, 1996.

[Cohen *et al.*, 1994] Philip R. Cohen, Adam J. Cheyer, M. Wang, and S. C. Baeg, "An Open Agent Architecture," In *AAAI Spring Symposium on Software Agents*, pages 1–8, March 1994.

[Cohen *et al.*, 1997] Philip R. Cohen, Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen, and Josh Clow, "Quickset: Multimodal interaction for distributed applications," In *Proceedings of the Fifth Annual International Multimodal Conference (Multimedia-97)*, pages 31–40. ACM Press, 1997.

[Emerson and Burstein, 1999] T. Emerson and Mark Burstein, "Development of a Constraint-based Airlift Scheduler by Program Synthesis from Formal Specifications," In *Proceedings of the Conference on Automated Software Engineering*. AAAI Press, 1999.

[Ferguson *et al.*, 1996a] George Ferguson, James Allen, and Brad Miller, "TRAINS-95: Towards a Mixed-Initiative Planning Assistant," In Brian Drabble, editor, *Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 70–77, Edinburgh, Scotland, 29–31 May 1996.

[Ferguson and Allen, 1998] George Ferguson and James F. Allen, "TRIPS: An Integrated Intelligent Problem-Solving Assistant," In *Proceedings of the Fifteenth National Conference on AI (AAAI-98)*, pages 567–573, Madison, WI, 26–30 July 1998.

[Ferguson *et al.*, 1996b] George Ferguson, James F. Allen, Brad W. Miller, and Eric K. Ringger, "The Design and Implementation of the TRAINS-96 System: A Prototype Mixed-Initiative Planning Assistant," TRAINS Technical Note 96-5, Department of Computer Science, University of Rochester, Rochester, NY, October 1996.

[Finin *et al.*, 1993] Tim Finin, Jay Weber, Gio Wiederhold, Michael Genesereth, Richard Fritzson, Donald McKay, James McGuire, Richard Pelavin, Stuart Shapiro, and Chris Beck, "Specification of the KQML Agent-Communication Language," Draft, 15 June 1993.

[Labrou and Finin, 1997] Yannis Labrou and Tim Finin, "A Proposal for a new KQML Specification," Technical Report CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1997.

[Sycara *et al.*, 1996] Katia Sycara, Keith Decker, Amandeep Pannu, Mike, Williamson, and Dajun Zeng, "Distributed Intelligent Agents," *IEEE Expert*, December 1996.

Appendix B: An Architecture for More Realistic Conversational Systems

Allen, J., G. Ferguson, and A. Stent (2001). An architecture for more realistic conversational systems. *Proceedings of the Conference on Intelligent User Interfaces (IUI-01)*, Santa Fe, NM, January 14-17, pages 1–8.

An Architecture For More Realistic Conversational Systems

James Allen, George Ferguson, Amanda Stent
Department of Computer Science
University of Rochester
Rochester, NY 14627-0226
{james,ferguson,stent}@cs.rochester.edu
http://www.cs.rochester.edu/research/trips/

ABSTRACT

In this paper, we describe an architecture for conversational systems that enables human-like performance along several important dimensions. First, interpretation is incremental, multi-level, and involves both general and task- and domain-specific knowledge. Second, generation is also incremental, proceeds in parallel with interpretation, and accounts for phenomena such as turn-taking, grounding and interruptions. Finally, the overall behavior of the system in the task at hand is determined by the (incremental) results of interpretation, the persistent goals and obligations of the system, and exogenous events of which it becomes aware. As a practical matter, the architecture supports a separation of responsibilities that enhances portability to new tasks and domains.

Keywords

Architectures for intelligent, cooperative, distributed, and multimodal interfaces; conversational systems

1. INTRODUCTION

Our goal is to design and build systems that approach human performance in conversational interaction. We limit our study to *practical dialogues*: dialogues in which the conversants are cooperatively pursuing specific goals or tasks. Applications involving practical dialogues include planning (e.g. designing a kitchen), information retrieval (e.g. finding out the weather), customer service (e.g. booking an airline flight), advice-giving (e.g. helping assemble modular furniture) or crisis management (e.g. a 911 center). In fact, the class of practical dialogues includes almost anything about which people might want to interact with a computer.

TRIPS, The Rochester Interactive Planning System [6], is an end-to-end system that can interact robustly and in near real-time using spoken language and other modalities. It has participated successfully in dialogues with untrained users in several different simple problem solving domains. Our experience building this system, however, revealed several problems that motivated the current work.

1.1 Incrementality

Like most other dialogue systems that have been built, TRIPS enforces strict turn taking between user and system, and processes each utterance sequentially through three stages: interpretation–dialogue management–generation. Unfortunately, these restrictions make the interaction unnatural and stilted, and will ultimately interfere with the user's ability to focus on the problem itself rather than on making the interaction work. We want an architecture that allows a more natural form of interaction; this requires incremental understanding and generation with flexible turn-taking.

Here are several examples of human conversation that illustrate some of the problems with processing in stages. All examples are taken from a corpus collected in an emergency management task set in Monroe County, NY [17]. Plus signs (+) denote simultaneous speech, and "␣" denotes silence.

First, in human-human conversation participants frequently ground (confirm their understanding of) each other's contributions using utterances such as "okay" and "mm-hm". Clearly, incremental understanding and generation are required if we are to capture this behavior. In the following example, A acknowledges each item in B's answers about locations where there are road outages.

Excerpt from Dialogue s16

A: can you give me the first uh ␣ outage
B: okay
B: so Elmwood bridge
A: okay
B: um ␣ Thurston road
A: mm-hm
B: + Three Eighty + Three at Brooks
A: + okay +
A: mm-hm
B: and Four Ninety at the inner ␣ loop
A: okay

Second, in human-human dialogues the responder frequently acknowledges the initiator's utterance immediately after it is completed and before they have performed the tasks they

IUI'01, January 14-17, 2001, Santa Fe, New Mexico, USA.

need to do to fully respond. In the next excerpt, A asks for problems other than road outages. B responds with an immediate acknowledgment. Evidence of problem solving activity is revealed by B smacking their lips ("lipsmack") and silence, and then B starts to respond to the request.

Excerpt from Dialogue s16

A: and what are the ␣ other um ␣ did you have just beside road $+_1$ outages $+_1$

B: $+_1$ okay $+_1$ <lipsmack> ␣ um Three Eighty Three and Brooks $+_2$ ␣ is $+_2$ a ␣ road out ␣ and an electric line down

A: $+_2$ Brooks mm hm $+_2$

A: okay

A sequential architecture, requiring interpretation and problem solving to be complete before generation begins, cannot produce this behavior in any principled way.

A third example involves interruptions, where the initiator starts to speak again after the responder has formulated a response and possibly started to produce it. In the example below, B starts to respond to A's initial statement but then A continues speaking.

Excerpt from Dialogue s6

A: and he's going to pull the tree

B: mm hm

B: and + there's mm +

A: + so ␣ he'll be + done at ␣ um ␣ he's going to be done at ␣ in forty minutes

We believe effective conversational systems are going to have to be able to interact in these ways, which are perfectly natural (in fact are the usual mode of operation) for humans. It may be that machines will not duplicate human behavior exactly, but they will realize the same conversational goals using the communication modalities they have. Rather than saying "uh-huh," for instance, the system might ground a referring expression by highlighting it on a display. Note also that the interruption example requires much more than a "barge-in" capability. B needs to interpret A's second utterance as a continuation of the first, and does not simply abandon its goal of responding to the first. When B gets the turn, B may decide to still respond in the same way, or to modify its response to account for new information.

1.2 Initiative

Another reason TRIPS does not currently support completely natural dialogue is that, like most other dialogue systems, it is quite limited in the form of mixed-initiative interaction it supports. It supports taking discourse-level initiative (cf. [3]) for clarifications and corrections, but does not allow shifting of task-level initiative during the interaction. The reason is that system behavior is driven by the dialogue manager, which focuses on interpreting user input. This means that the system's own independent goals are deemphasized. The behavior of a conversational agent should ideally be determined by three factors, not just one: the interpretation of the last user utterance (if any), the agent's own persistent goals and obligations, and exogenous events of which it becomes aware.

For instance, in the Monroe domain, one person often chooses to ignore the other person's last utterance and leads the conversation to discuss some other issue. Sometimes they explicitly acknowledge the other's contribution and promise to address it later, as in the following:

Excerpt from Dialogue s16

A: can you ␣ $+_1$ can $+_1$ you go over the ␣ the thing $+_2$ for me again $+_2$

B: $+_1$ i $+_1$

B: $+_2$ yeah in one $+_2$ minute

B: i have to ␣ clarify at the end here ....

In other cases, they simply address some issue they apparently think is more important than following the other's lead, as in the following example where B does not address A's suggestion about using helicopters in any explicit way.

Excerpt from Dialogue s12

A: we can we ␣ we can either uh

A : i guess we have to decide how to break up ␣ this

A: we can ␣ make three trips with a helicopter

B: so i guess we should send one ambulance straight off ␣ to ␣ marketplace right ␣ now ␣ right

1.3 Portability

Finally, on a practical note, while TRIPS was designed to separate discourse interpretation from task and domain reasoning, in practice domain- and task-specific knowledge ended up being used directly in the dialogue manager. This made it more difficult to port the system to different domains and also hid the difference between general domain-independent discourse behavior and task-specific behavior in a particular domain.

To address these problems, we have developed a new architecture for the "core" of our conversational system that involves asynchronous interpretation, generation, and system planning/acting processes. This design simplifies the incremental development of new conversational behaviors. In addition, our architecture has a clean separation between discourse modeling and task/domain levels of reasoning, which (a) enhances our ability to handle more complex domains, (b) improves portability between domains; and (c) allows for richer forms of task-level initiative.

The remainder of this paper describes our new architecture in detail. The next section presents an overview of the design and detailed descriptions of the major components. A brief but detailed example illustrates the architecture in action. We conclude with a discussion of related work on conversational systems and the current status of our implementation.

2. ARCHITECTURE DESCRIPTION

As mentioned previously, we have been developing conversational agents for some years as part of the TRAINS [7] and TRIPS [6] projects. TRIPS is designed as a loosely-coupled collection of components that exchange information by passing messages. There are components for speech processing (both recognition and synthesis), language understanding, dialogue management, problem solving, and so on.

In previous versions of the TRIPS system, the Dialogue Manager component (DM) performed several functions:

- Interpretation of user input in context

- Maintenance of discourse context
- Planning the content (but not the form) of system responses
- Managing problem solving and planning

Having all these functions performed by one component led to several disadvantages. The distinction between domain planning and discourse planning was obscured. It became difficult to improve interpretation and response planning, because the two were so closely knit. Incremental processing was difficult to achieve, because all input had to pass through the DM (even if no domain reasoning was going to occur, but only discourse planning). Finally, porting the system to new tasks and domains was hampered by the interconnections between the various types of knowledge within the DM.

The new core architecture of TRIPS is shown in Figure 1. There are three main processing components. The Interpretation Manager (IM) interprets user input as it arises. It broadcasts the recognized speech acts and their interpretation as problem solving actions, and incrementally updates the Discourse Context. The Behavioral Agent (BA) is most closely related to the autonomous "heart" of the agent. It plans system behavior based on its goals and obligations, the user's utterances and actions, and changes in the world state. Actions that involve communication and collaboration with the user are sent to the Generation Manager (GM). The GM plans the specific content of utterances and display updates. Its behavior is driven by discourse obligations (from the Discourse Context), and directives it receives from the BA. The glue between the layers is an abstract model of problem solving in which both user and system contributions to the collaborative task can be expressed.

All three components operate asynchronously. For instance, the GM might be generating an acknowledgment while the BA is still deciding what to do. And if the user starts speaking again, the IM will start interpreting these new actions. The Discourse Context maintains the shared state needed to coordinate interpretation and generation.

In the remainder of this section, we describe the major components in more detail, including descriptions of the Discourse Context, Problem Solving Model, and Task Manager.

2.1 Discourse Context

The TRIPS Discourse Context provides information to coordinate the system's conversational behavior. First, it supplies sufficient information to generate and interpret anaphoric expressions and to interpret forms of ellipsis. Given the real-time nature of the interactions, and the fact that the system may have its own goals and receive reports about external events, the discourse context must also provide information about the status of the turn (i.e. can I speak now or should I wait?), and what discourse obligations are currently outstanding (cf. [19]). The latter is especially important when the system chooses to pursue some other goal (e.g. notifying the user of an accident) rather than perform the expected dialogue act (e.g. answering a question); to be coherent and cooperative, the system should usually still satisfy outstanding discourse obligations, even if this is done simply by means of an apology. Finally, as we move towards open-mike interactive systems, we must also identify and generate appropriate grounding behaviors. To support these needs, the TRIPS discourse context contains the following information:

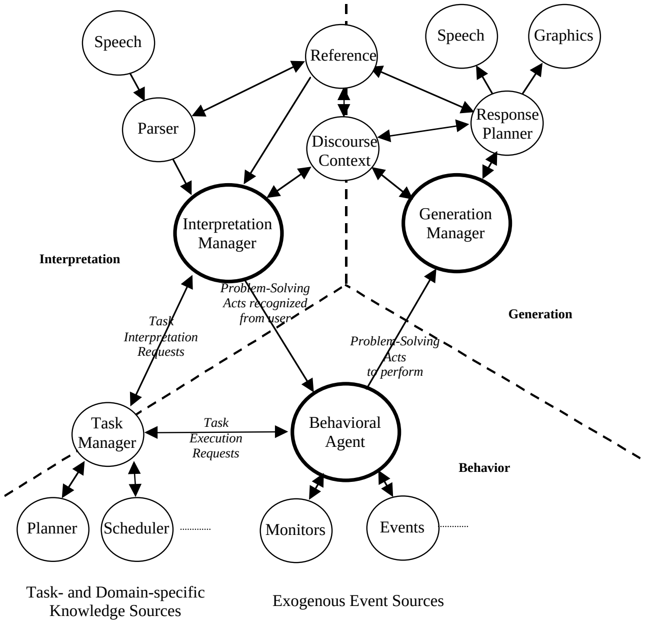


Figure 1: New Core Architecture

1. A model of the current salient entities in the discourse, to support interpretation and generation of anaphoric expressions;
2. The structure and interpretation of the immediately preceding utterance, to support ellipsis resolution and clarification questions;
3. The current status of the turn—whether it is assigned to one conversant or currently open.
4. A discourse history consisting of the speech-act interpretations of the utterances in the conversation so far, together with an indication of which utterances have been grounded;
5. The current discourse obligations, typically to respond to the other conversant's last utterance. Obligations may act as a stack during clarification subdialogues, or short-term interruptions, but this stack never becomes very large.

This is a richer discourse model than found in most systems (although see [12] for a model of similar richness).

2.2 Abstract Problem Solving Model

The core modules of the conversational agent, the IM, BA and GM, use general models of collaborative problem solving, but these models remain at an abstract level, common to all practical dialogues. This model is formalized as a set of actions that can be performed on problem solving objects. The problem solving objects include *objectives* (goals being pursued), *solutions* (proposed courses of action or structures that may achieve an objective), *resources* (objects used in solutions, such as trucks for transportation, space in kitchen design), and *situations* (settings in which solutions are used to attain objectives).

In general, there are a number of different actions agents can perform as they collaboratively solve problems. Many of these can apply to any problem solving object. For example, agents may *create* new objectives, new solutions, new situations (for hypothetical reasoning) and new resources (for resource planning). Other actions in our abstract problem solving model include *select* (e.g.focus on a particular objective), *evaluate* (e.g. determine how long a solution might take), *compare* (e.g. compare two solutions to the same objective), *modify* (e.g. change some aspect of a solution, change what resources are available), *repair* (e.g. fix an old solution so that it works) and *abandon* (e.g. give up on an objective, throw out a possible solution)[1]. Because we are dealing with collaborative problem solving, not all of these actions can be accomplished by one agent alone. Rather, one agent needs to propose an action (the agent is said to *initiate* the collaborative act), and the other accept it (the other agent *completes* the collaborative act).

There are also explicit communication acts involved in collaborative problem solving. Like all communicative acts, these acts are performed by a single agent, but are only successful if the other agent understands the communication. The main communication acts for problem solving include *describe* (e.g. elaborate on an objective, describe a particular solution), *explain* (e.g. provide a rationale for a solution or decision), and *identify* (e.g. communicate the existence of a resource, select a goal to work on). These communication acts, of course, may be used to accomplish other problem solving goals as well. For instance, one might initiate the creation of an objective by describing it.

2.3 Task Manager

The behaviors of the IM, BA and GM are defined in terms of the abstract problem solving model. The details of what these objects are in a particular domain, and how operations are performed, are specified in the Task Manager (TM). The TM supports operations intended to assist in both the *recognition* of what the user is doing with respect to the task at hand and the *execution* of problem solving steps intended to further progress on the task at hand.

Specifically, the Task Manager must be able to:

1. Answer queries about objects and their role in the task/domain (e.g. is an ambulance a resource? Is loading a truck an in-domain plannable/executable action? Is "evacuating a city" a possible in-domain goal?)

2. Provide the interface between the generic problem solving acts used by the BA (e.g. create a solution) and the actual task-specific agents that perform the tasks (e.g. build a course of action to evacuate the city using two trucks)

3. Provide intention recognition services to the IM (e.g. can "going to Avon" plausibly be an extension of the current course of action?)

In our architecture, the Task Manager maps abstract problem solving acts onto the capabilities of the knowledge-based agents at its disposal. For example, in one of our planning domains, the Task Manager uses a planner, router, scheduler, and temporal knowledge base to answer queries and create or modify plans.

2.4 Interpretation Manager

The Interpretation Manager (IM) interprets incoming parsed utterances and generates updates to the Discourse Context. First, it produces turn-taking information. With a push-to-talk interface this is simple. When the user presses the button they have taken the turn; when they release it they have released the turn. As we move to open-mike, identifying turn-taking behavior will require more sophisticated interpretation. TRIPS uses an incremental chart parser that will assist in this process by broadcasting constituents as they are recognized.

The principal task of the IM, however, is to identify the intended speech act, the collaborative problem solving act that it furthers, and the system's obligations arising from the interaction. For instance, the utterance "The bridge over the Genesee is blocked" would be interpreted in some circumstances as a problem statement, the intention being to initiate replanning. The IM would broadcast a discourse-level obligation to respond to a statement, and announce that the user has initiated the collaborative problem solving act of identifying a problem as a means of initiating replanning (say, to change the route currently planned). In other circumstances, the same utterance might be recognized as the introduction of a new goal (i.e. to reopen the bridge). The rules to construct these interpretations are based on the abstract problem solving model and specific decisions are made by querying the Task Manager. For instance, in the above example, key questions might be "is there an existing plan using the bridge?" (an affirmative answer indicates the replanning interpretation) and "is making the bridge available a reasonable high-level goal in this domain?" (an affirmative answer indicates the introduce-goal interpretation).

2.5 Generation Manager

The Generation Manager (GM), which performs content planning, receives problem solving goals requiring generation from the Behavioral Agent (BA) and discourse obligations from the Discourse Context. The GM's task is to synthesize these input sources and produce plans (sequences of discourse acts) for the system's discourse contributions.

Because the GM operates asynchronously from the IM, it can be continuously planning. For instance, it is informed when the user's turn ends and can plan simple take-turn and keep-turn acts even in the absence of further information from the IM or the BA, using timing information.

In the case of grounding behaviors and some conventional interactions (e.g. greetings), the GM uses simple rules based on adjacency pairs; no reference to the problem solving state is necessary. In other cases, it may need information from the BA in order to satisfy a discourse obligation. It may also receive goals from the Behavioral Agent that it can plan to satisfy even in the absence of a discourse obligation, for instance when something important changes in the world and the BA wants to notify the user.

The GM can also plan more extensive discourse contributions using rhetorical relations expressed as schemas, for instance to explain a fact or proposal or to motivate a proposed action. It has access to the discourse context as well

[1] This list is not meant to be exhaustive, although it has been developed based on our experiences building systems in several problem solving domains.

as to sources for task- and domain-level knowledge.

When the GM has constructed a discourse act or set of acts for production, it sends the act(s) and associated content to the Response Planner, which performs surface generation. The RP comprises several subcomponents; some are template-based, some use a TAG-based grammar, and one performs output selection and coordination. It can realize turn-taking, grounding and speech acts in parallel and in real-time, employing different modalities where useful. It can produce incremental output at two levels: it can produce the output for one speech act before others in a plan are realized; and where there is propositional content, it can produce incremental output within the sentence (cf. [10]). If a discourse act is realized and produced successfully, the GM is informed and sends an update to the Discourse Context.

2.6 Behavioral Agent

As described above, the Behavioral Agent (BA) is responsible for the overall problem solving behavior of the system. This behavior is a function of three aspects of the BA's environment: (1) the interpretation of user utterances and actions in terms of problem solving acts, as produced by the Interpretation Manager; (2) the persistent goals and obligations of the system, in terms of furthering the problem solving task; (3) Exogenous events of which the BA becomes aware, perhaps by means of other agents monitoring the state of the world or performing actions on the BA's behalf.

As we noted previously, most dialogue systems (including previous versions of TRIPS) respond primarily to the first of these sources of input, namely the user's utterances. In some systems (including previous versions of TRIPS) there is some notion of the persistent goals and/or obligations of the system. Often this is implicit and "hard-coded" into the rules governing the behavior of the system. In realistic conversational systems, however, these would take on a much more central role. Just as people do, the system must juggle its various needs and obligations and be able to talk about them explicitly.

Finally, we think it is crucial that conversational systems get out into the world. Rather than simply looking up answers in a database or even conducting web queries, a conversational system helping a user with a real-world task is truly an agent embedded in the world. Events occur that are both exogenous (beyond its control) and asynchronous (occurring at unpredictable times). The system must take account of these events and integrate them into the conversation. Indeed in many real-world tasks, this "monitoring" function constitutes a significant part of the system's role.

The Behavioral Agent operates by reacting to incoming events and managing its persistent goals and obligations. In the case of user-initiated problem solving acts, the BA determines whether to be cooperative and how much initiative to take in solving the joint problem. For example, if the user initiates creating a new objective, the system can complete the act by adopting a new problem solving obligation to find a solution. It could, however, take more initiative, get the Task Manager to compute a solution (perhaps a partial or tentative one), and further the problem solving by proposing the solution to the user.

The BA also receives notification about events in the world and chooses whether to communicate them to the user and/or adopt problem solving obligations about them. For example, if the system receives a report of a heart attack victim needing attention, it can choose to simply inform the user of this fact (and let them decide what to do about it). More likely, it can decide that something should be done about the situation, and so adopt the intention to solve the problem (i.e. get the victim to a hospital).

Thus the system's task-level initiative-taking behavior is determined by the BA, based on the relative priorities of its goals and obligations. These problem-solving obligations determine how the system will respond to new events, including interpretations of user input.

2.7 Infrastructure

The architecture described in this paper is built on an extensive infrastructure that we have developed to support effective communication between the various components making up the conversational system. Space precludes an extended discussion of these facilities, but see [1] for further details.

System components communicate using the Knowledge Query and Manipulation Language (KQML [11]), which provides a syntax and high-level semantics for messages exchanged between agents. KQML message traffic is mediated by a Facilitator that sits at the hub of a star topology network of components. While a hub may seem to be a bottleneck, in practice this has not been a problem. On the contrary, the Facilitator provides a variety of services that have proven indispensable to the design and development of the overall system. These include: robust initialization, KQML message validation, naming and lookup services, broadcast facilities, subscription (clients can subscribe in order to receive messages sent by other clients), and advertisement (clients may advertise their capabilities).

The bottom line is that an architecture for conversational systems such as the one we are proposing in this paper would be impractical, if not impossible, without extensive infrastructure support. While these may seem like "just implementation details," in fact the power and flexibility of the TRIPS infrastructure enables us to design the architecture to meet the needs of realistic conversation *and to make it work.*

3. EXAMPLE

An example will help clarify the relationships between the various components of our architecture and the information that flows between them, as well as the necessity for each.

Consider the situation in which the user asks "Where are the ambulances?" First, the speech recognition components notice that the user has started speaking. This is interpreted by the Interpretation Manager as taking the turn, so it indicates that a TAKE-TURN event has occurred:

```
(tell (done (take-turn :who user)))
```

The Generation Manager might use this information to cancel or delay a planned response to a previous utterance. It can also be used to generate various grounding behaviors (e.g. changing a facial expression, if such a capability is supported). When the utterance is completed, the IM interprets the user's having stopped speaking as releasing the turn:

```
(tell (done (release-turn :who user)))
```

At this point, the GM may start planning (or executing) an appropriate response.

The Interpretation Manager also receives a logical form describing the surface structure of this request for information. It performs interpretation in context, interacting with the Task Manager. In this case, it asks the Task Manager if ambulances are considered resources in this domain. With an affirmative response, it interprets this question as initiating the problem solving act of identifying relevant resources. Note that contextual interpretation is critical—the user wants to know where the usable ambulances are, not where all known ambulances might be. The IM then generates:

1. A message to the Discourse Context recording the user's utterance in the discourse history together with its structural analysis from the parser.

2. A message to the Discourse Context that the system now has an obligation to respond to the question:

   ```
   (tell
    (introduce-obligation
     :id OBLIG1
     :who system
     :what (respond-to
            (wh-question
             :id UTT1
             :who user
             :what (at-loc (the-set ?x
                             (type ?x ambulance))
                           (wh-term ?l
                             (type ?l location)))
             :why (initiate PS1)))))
   ```

   This message includes the system's obligation, a representation of the content of the question, and a connection to the recognized problem solving act (defined in the message described next). The IM does not specify how the obligation to respond to the question should be discharged.

3. A message to the Behavioral Agent that the user has initiated a collaborative problem solving act, namely attempting to identify a resource:

   ```
   (tell
    (done
     (initiate
      :who user
      :what (identify-resource
             :id PS1
             :what (set-of ?x
                     (type ?x ambulance))))))
   ```

   This message includes the problem solving act recognized by the IM as the user's intention, and a representation of the content of the question.

When the Discourse Context receives notification of the new discourse obligation, this fact is broadcast to any subscribed components, including the Generation Manager. The GM cannot answer the question without getting a response from the Behavioral Agent. So it adopts the goal of answering, and waits for information from the BA. While waiting, it may plan and produce an acknowledgment of the question.

When the Behavioral Agent receives notification that the user has initiated a problem solving act, one of four things can happen depending on the situation. We will consider each one in sequence.

Do the Right Thing It may decide to "do its part" and try to complete (or at least further) the problem solving. In this case, it would communicate with other components to answer the query about the location of the ambulances, and then send the GM a message like:

```
(request
 (identify-resource
  :who system
  :what (and
         (at-loc amb-1 rochester)
         ...)
  :why (complete :who system :what PS1)))
```

The BA expects that this will satisfy its problem solving goal of completing the identify-resources act initiated by the user, although it can't be sure until it hears back from the IM that the user understood the response.

Clarification The BA may try to identify the resource but fail to do so. If a specific problem can be identified as having caused the failure, then it could decide to initiate a clarification to obtain the information needed. For instance, say the dialogue has so far concerned a particular subtask involving a particular type of ambulances. It might be that the BA cannot decide if it should identify just the ambulances of the type for this subtask, or whether the user wants to know where all usable ambulances are. So it might choose to tell the GM to request a clarification. In this case, the BA retains its obligation to perform the identify-resources act.

Failure On the other hand, the BA may simply fail to identify the resources that the user needs. For instance, the agents that it uses to answer may not be responding, or it may be that the question cannot be answered. In this case, it requests the GM to notify the user of failure, and abandons (at least temporarily) its problem solving obligation.

Ignoring the Question Finally, the BA might decide that some other information is more important, and send that information to the GM (e.g. if a report from the world indicates a new and more urgent task for the user and system to respond to). In this case, the BA retains the obligation to work on the pending problem solving action, and will return to it when circumstances permit.

Whatever the situation, the Generation Manager receives some abstract problem solving act to perform. It then needs to reconcile this act with its discourse obligation OBLIG1. Of course, it can satisfy OBLIG1 by answering the question. It can also satisfy OBLIG1 by generating a clarification request, since the clarification request is a satisfactory response to the question. (Note that the obligation to answer the original question is maintained as a problem solving goal, not a discourse obligation). In the case of a failure, OBLIG1 could be satisfied by generating an apology and a description of the reason the request could not be satisfied. If the BA ignores the question, the GM might apologize and add a promise to address the issue later, before producing the unrelated information. The apology would satisfy OBLIG1. For a very urgent message (e.g. a time critical warning), it might generate the warning immediately, leaving the discourse obligation OBLIG1 unsatisfied, at least temporarily.

The GM sends discourse acts with associated content to the Response Planner, which produces prosodically-annotated text for speech synthesis together with multimodal display commands. When these have been successfully (or partially in the case of a user interruption) produced, the GM is informed and notifies the Discourse Context as to which discourse obligations should have been met. It also gives the Discourse Context any expected user obligations that result from the system's utterances.

The Interpretation Manager uses knowledge of these expectations to aid subsequent interpretation. For example, if an answer to the user's question is successfully produced, then the user has an obligation to acknowledge the answer. Upon receiving an acknowledgment (or inferring an implicit acknowledge), the IM notifies the Discourse Context that the obligation to respond to the question has truly been discharged, and might notify the BA that the collaborative "Identify-Resource" act PS1 has been completed.

4. IMPLEMENTATION

The architecture described in this paper arose from a long-term effort in building spoken dialogue systems. Because we have been able to easily port most components from our previous system into the new one, the system itself has a wide range of capabilities that were already present in earlier versions. Specifically, it handles robust, near real-time spontaneous dialogue with untrained users as they solve simple tasks such as trying to find routes on a train map and planning evacuation of personnel from an island (see [1] for an overview of the different domains we have implemented). The system supports cooperative, incremental development of plans with clarifications, corrections, modifications and comparison of different options, using unrestricted, natural language (as long as the user stays focussed on the task at hand). The new architecture extends our capabilities to better handle the incremental nature of interpretation, the fact that interpretation and generation must be interleaved, and the fact that realistic dialogue systems must also be part of a broader outside world that is not static. The new architecture further clarifies the separation between linguistic and discourse knowledge on one hand, and task and domain knowledge on the other.

We demonstrated an initial implementation of our new architecture in August 2000, providing the dialogue capabilities for an emergency relief planning domain which used simulation, scheduling, and planning components built by research groups at other institutions. Current work involves extending the capabilities of individual components (the BA and GM in particular) and porting the system to a more complex emergency-handling domain [17].

5. RELATED WORK

Dialogue systems are now in use in many applications. Due to space constraints, we have selected only some of these for comparison to our work. They cover a range of domains, modalities and dialogue management types:

- Information-seeking systems [2, 5, 8, 9, 13, 15, 16] and planning systems [4, 14, 18];
- Speech systems [13, 14, 15, 16], multi-modal systems [5, 8, 18] and embodied conversational agents [2, 9];
- Systems that use schemas or frames to manage the dialogue [9, 13, 14, 16], ones that use planning [4], ones that use models of rational interaction [15], and ones that use dialogue grammars or finite state models [5, 8, 18].

Most of the systems we looked at use a standard interpretation–dialogue management–generation core, with the architecture being either a pipeline or organized around a message-passing hub with a pipeline-like information flow. Our architecture uses a more fluid processing model, which enables the differences we outline below.

5.1 Separation of domain/task reasoning from discourse reasoning

Since many dialogue systems are information-retrieval systems, there may be fairly little task reasoning to perform. For that reason, although many of these systems have domain models or databases separate from the dialogue manager [5, 8, 9, 13, 15, 16], they do not have separate task models. By contrast, our system is designed to be used in domains such as planning, monitoring, and design, where task-level reasoning is crucial not just for performing the task but also for interpreting the user's utterances. Separation of domain knowledge and task reasoning from discourse reasoning – through the use of our Task Manager, various world models, the abstract problem solving model and the Behavioral Agent – allows us access to this information without compromising portability and flexibility.

CommandTalk [18], because it is a thin layer over a standalone planner-simulator, has little direct involvement in task reasoning. However, the dialogue manager incorporates some domain-dependent task reasoning, e.g. in the discourse states for certain structured form-filling dialogues.

In the work of Cassell et al [2], the response planner performs deliberative task and discourse reasoning to achieve communicative and task-related goals. In our architecture, there is a separation between task- and discourse-level planning, with the Behavioral Agent handling the first type of goal and the Generation Manager the other.

Chu-Carroll and Carberry's CORE [4] is not a complete system, but does have a specification for input to the response planner that presumably would come from a dialogue manager. The input specification allows for domain, problem solving, belief and discourse-level intentions. Our Interpretation and Generation Managers reason over discourse-level intentions; they obtain information about domain, problem solving and belief intentions from other modules.

The CMU Communicator system has a dialogue manager, but uses a set of domain agents to "handle all domain-specific information access and interpretation, with the goal of excluding such computation from the dialogue management component" [14]. However, the dialogue manager uses task- or domain-dependent schemas to determine its behavior.

5.2 Separation of interpretation from response-planning

Almost all the systems we examined combine interpretation with response planning in the dialogue manager. The architecture outlined by Cassell et al [2], however, separates the two. It includes an understanding module (performing the same kinds of processing performed by our Interpretation Manager); a response planner (performing deliberative reasoning); and a reaction module (which performs action coordination and handles reactive behaviors such as turn-taking). We do not have a separate component to process reactive behaviors; we get reactive behaviors because different types of goals take different paths through our system. Cassell et al's "interactional" goals (e.g. turn-taking, grounding) are handled completely by the discourse components of our system (the Interpretation and Generation Managers); the handling of their "propositional" goals may involve domain or task reasoning and therefore will involve our Behavioral Agent and problem-solving modules.

Fujisaki et al [8] divide discourse processing into a user model and a system model. As in other work [4, 15], this is an attempt to model the beliefs and knowledge of the agents participating in the discourse, rather than the discourse itself. However, interpretation must still be completed before response planning begins. Furthermore, the models of user and system are finite-state models; for general conversational agents more flexible models may be necessary.

6. CONCLUSIONS

We have described an architecture for the design and implementation of conversational systems that participate effectively in realistic practical dialogues. We have emphasized the fact that interpretation and generation must be interleaved and the fact that dialogue systems in realistic settings must be part of and respond to a broader "world outside." These considerations have led us to an architecture in which interpretation, generation, and system behavior are functions of autonomous components that exchange information about both the discourse and the task at hand. A clean separation between linguistic and discourse knowledge on the one hand, and task- and domain-specific information on the other hand, both clarifies the roles of the individual components and improves portability to new tasks and domains.

7. ACKNOWLEDGMENTS

This work is supported by ONR grant no. N00014-95-1-1088, DARPA grant no. F30602-98-2-0133, and NSF grant no. IRI-9711009.

8. REFERENCES

[1] J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. An architecture for a generic dialogue shell. *J. Natural Language Engineering*, 6(3):1–16, 2000.

[2] J. Cassell, T. Bickmore, L. Campbell, K. Chang, H. Vilhjálmsson, and H. Yan. Requirements for an architecture for embodied conversational characters. In D. Thalmann and N. Thalmann, editors, *Proceedings of Computer Animation and Simulation '99*, 1999.

[3] J. Chu-Carroll and M. Brown. Initiative in collaborative interactions – its cues and effects. In *Proceedings of AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, 1997.

[4] J. Chu-Carroll and S. Carberry. Collaborative response generation in planning dialogues. *Computational Linguistics*, 24(3):355–400, 1998.

[5] N. Dahlbäck, A. Flycht-Eriksson, A. Jönsson, and P. Qvarfordt. An architecture for multi-modal natural dialogue systems. In *Proceedings of ESCA Tutorial and Research Workshop (ETRW) on Interactive Dialogue in Multi-Modal Systems*, 1999.

[6] George Ferguson and James F. Allen. TRIPS: An integrated intelligent problem-solving assistant. In *Proceedings of AAAI-98*, pages 567–573, 1998.

[7] George Ferguson, James F. Allen, Brad W. Miller, and Eric K. Ringger. The design and implementation of the TRAINS-96 system: A prototype mixed-initiative planning assistant. TRAINS Technical Note 96-5, Department of Computer Science, University of Rochester, 1996.

[8] H. Fujisaki, H. Kameda, S. Ohno, K. Abe, M. Iijima, M. Suzuki, and Z. Taketa. Principles and design of an intelligent system for information retrieval over the internet with a multimodal dialogue interface. In *Proceedings of Eurospeech'99*, 1999.

[9] J. Gustafson, N. Lindberg, and M. Lundeberg. The August spoken dialogue system. In *Proceedings of Eurospeech'99*, 1999.

[10] A. Kilger and W. Finkler. Incremental generation for real-time applications. Technical Report RR-95-11, Deutsches Forschungzentrum für Künstliche Intelligenz GmbH (DFKI), Saarbrücken, Germany, 1995.

[11] Yannis Labrou and Tim Finin. A proposal for a new KQML specification. Technical Report CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1997.

[12] C. Matheson, M. Poesio, and D. Traum. Modelling grounding and discourse obligations using update rules. In *Proceedings of NAACL-2000*, 2000.

[13] S. Rosset, S. Bennacef, and L. Lamel. Design strategies for spoken dialog systems. In *Proceedings of Eurospeech'99*, 1999.

[14] A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh. Creating natural dialogs in the Carnegie Mellon Communicator system. In *Proceedings of Eurospeech'99*, 1999.

[15] M. Sadek, P. Bretier, and F. Panaget. ARTIMIS: Natural dialogue meets rational agency. In *Proceedings of IJCAI-97*, 1997.

[16] S. Seneff and J. Polifroni. Dialogue management in the Mercury flight reservation system. In *Proceedings of ANLP-NAACL2000 Workshop on Conversational Systems*, pages 11–16, 2000.

[17] A. Stent. The Monroe corpus. Technical Report 728, Department of Computer Science, University of Rochester, March 2000.

[18] A. Stent, J. Dowding, J. Gawron, E. Owen Bratt, and R. Moore. The CommandTalk spoken dialogue system. In *Proceedings of ACL-99*, pages 183–190, 1999.

[19] D. Traum and J. Allen. Discourse obligations in dialogue processing. In *Proceedings of ACL-94*, pages 1–8, 1994.

Appendix C: A Problem Solving Model for Collaborative Agents

Allen, J., N. Blaylock, and G. Ferguson (2002). A Problem Solving Model for Collaborative Agents. *Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, Bologna, Italy, July 31–August 2.

A Problem Solving Model for Collaborative Agents

James Allen
Dept. of Computer Science
University of Rochester
Rochester, New York, USA
james@cs.rochester.edu

Nate Blaylock
Dept. of Computer Science
University of Rochester
Rochester, New York, USA
blaylock@cs.rochester.edu

George Ferguson
Dept. of Computer Science
University of Rochester
Rochester, New York, USA
ferguson@cs.rochester.edu

ABSTRACT

This paper describes a model of problem solving for use in collaborative agents. It is intended as a practical model for use in implemented systems, rather than a study of the theoretical underpinnings of collaborative action. The model is based on our experience in building a series of interactive systems in different domains, including route planning, emergency management, and medical advising. It is currently being used in an implemented, end-to- end spoken dialogue system in which the system assists a person in managing their medications. While we are primarily focussed on human-machine collaboration, we believe that the model will equally well apply to interactions between sophisticated software agents that need to coordinate their activities.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Natural Language*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Theory and Methods*

Keywords

Conversational agents, interface agents, coordinating multiple agents and activities, intention recognition

1. INTRODUCTION

One of the most general models for interaction between humans and autonomous agents is based on natural human-human dialogue. For humans, this is an interface that requires no learning, and provides maximum flexibility and generality. To build such an interface on the autonomous agent side, however, is a formidable undertaking. We have been building prototypes of such systems for many years, focusing on limited problem solving tasks. Our approach involves constructing a dialogue system that serves as the interface between the human and the back-end agents. The goal is to insulate the human from the complexities of managing and understanding agent-based systems, while insulating the back-end agents from having to understanding natural language dialogue. To be effective in a range of situations, the dialogue agent must support contextually-dependent interpretation of language and be able to map linguistically specified goals into concrete tasking of back-end agents.

We believe that a key for enabling such interaction models is the development of a rich model of collaborative problem solving. This model is needed for two distinct purposes: (1) to enable contextual interpretation of language (*i.e.*, intention recognition); and (2) to provide a rich protocol for communication between the autonomous agents that comprise the dialogue system. Thus the dialogue system appears to the human as an intelligent collaborative assistant agent, and is itself comprised of autonomous agents.

While work has been done on general theoretical frameworks for collaborative interaction [8, 3, 11], these proposals have generally not specified the details of what such models would look like. We believe that our model is compatible with the SharedPlans formalism [8, 9, 11]. In fact, one way of looking at our model is as an elaboration of some of the key operators (such as Elaborate_Group, or Lochbaum's *communicate* recipe) in the SharedPlans framework. In our own previous work [6, 1], we have described the beginnings of practical models but these have not been very precisely specified or complete. In this paper, we sketch a comprehensive model that provides a detailed analysis of a wide range of collaborative problem solving situations that can arise. This model is based on our experience in building collaborative problem solving agents in a range of different domains. In particular, collaborative agents (both human and autonomous) need to have the capability to:

1. Discuss and negotiate goals;
2. Discuss options and decide on courses of action, including assigning different parts of a task to different agents;
3. Discuss limitations and problems with the current course of action, and negotiate modifications;
4. Assess the current situation and explore possible future eventualities;
5. Discuss and determine resource allocation;
6. Discuss and negotiate initiative in the interactions;

AAMAS'02, July 15-19, 2002, Bologna, Italy.
Copyright 2002 ACM 1-58113-480-0/02/0007 ...$5.00.

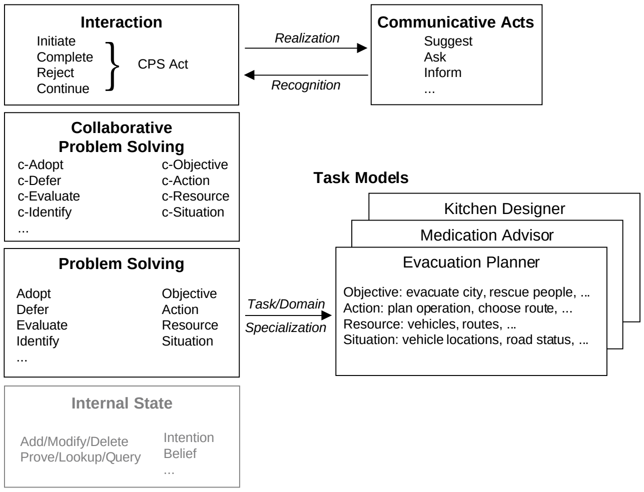


Figure 1: Collaborative problem solving model

7. Perform parts of the task, and report to others to update shared knowledge of the situation.

Although our focus is on language-based interaction, it is our belief that these capabilities are required in any sufficiently complex (realistic, flexible) agent-based system.

2. OVERVIEW OF THE MODEL

Our model of collaborative problem solving is shown in Figure 1. At the heart of the model is the **problem solving level**, which describes how a single agent solves problems. For example, an agent might adopt an obligation, or might evaluate the likelihood that a certain action will achieve that objective. This level is based on a fairly standard model of agent behavior, that we will describe in more detail shortly.[1]

The problem solving level is specialized to a particular task and domain by a **task model**. The types of domains we have explored include designing a kitchen, providing medical advice, assessing damage from a natural disaster, planning emergency services, and so on. The task model describes how to perform these tasks, such as what possible objectives are, how objectives are (or might be) related, what resources are available, and how to perform specific problem solving actions such as evaluating a course of action.

For an isolated autonomous agent, these two levels suffice to describe its behavior, including the planning and execution of task-level actions. For collaborative activity, however, we need more.

The **collaborative problem solving level** builds on the single-agent problem solving level. The collaborative problem solving actions parallel the single-agent ones, except that they are joint actions involving jointly understood objects. For example, the agents can jointly adopt an intention (making it a joint intention), or they can jointly identify a relevant resource, and so on.

Finally, an agent cannot simply perform a collaborative action by itself. The **interaction level** consists of actions performed by individuals in order to perform *their* part of collaborative problem solving acts. Thus, for example, one agent may initiate a collaborative act to adopt a joint intention, and another may complete the collaborative act by agreeing to adopt the intention.

This paper proceeds as follows. First, we describe the collaborative problem solving model in more detail, starting with a review of some underlying concepts, moving on to the single-agent problem solving level, and finally describing the collaborative problem solving and interaction levels. The emphasis is on the information maintained at each level and its use during collaborative problem solving. We then present a detailed example of the model in action, drawn from a medical advisor domain that we are using for our prototype implementation. We conclude with a few comments about open issues and future work.

3. BASIC CONCEPTS

All of the levels in our model involve a core set of concepts related to planning and acting. Many of these concepts have been used in the planning literature for years, and we only informally describe them in this section. The application of the concepts to modeling collaborative interaction is what is important for present purposes.

3.1 Situations

We start with a fairly standard notion of **situation** as in the situation calculus [12]—a situation is a snapshot of the world at a particular point in time (or hypothetical point in time when planning into the future). While situations are a complete state of the world at a certain time, our knowledge of a situation is necessarily incomplete except in the most simple cases (like traditional blocks world planning systems). Also note that a situation might include an agent's beliefs about the past and the future, and so might entail knowledge about the world far beyond what is immediately true.

3.2 Atomic Actions

Also as in the situation calculus, **actions** are formalized as functions from one situation to another. Thus, performing an action in one situation produces a new situation. Of course, generally we do not know the actual situation we are in, so typically knowledge about actions is characterized by statements that if some precondition of an action is true in some situation, then some effect of it will be true in the situation resulting from the action. Note that unlike the standard situation calculus, however, we take actions to be extended in time and allow complex simultaneous and overlapping actions.

3.3 Recipes

A specification of system behavior is often called a plan, or **recipe** [13]. We will use the term "recipe" here as the notion of plan has been overused and so is ambiguous. A very simple form of recipe is a fixed sequence of actions to perform, much like those built by traditional planning systems. The recipes found in cookbooks often aspire to this level of simplicity but typically are not as straightforward. More generally, recipes capture complex learned behavior and guide an agent towards a goal through a wide range

[1] Underlying the problem solving level is the representation of the agent's internal state, for example its current beliefs and intentions. The details of how these are represented are not important for understanding the collaborative problem solving model, however.

of possible conditions and ranges of possible results from previous action.

For our work, we do not care about the specific form of what a recipe is, or insist that different agents have the same recipes. Rather, a recipe is a way of deciding what to do next. More formally, a recipe is a function from situations to actions, where the action is the next thing to do according to the recipe.

Note that we need some special "actions" to make this work. First, we must allow the action of doing nothing or waiting for some period of time, as this might be the best thing to do for some recipes. We also need to allow the possibility that a recipe may not specify what to do next in certain situations. To formalize this, we need to make the recipe function a partial function, or introduce a special "failure" value. Finally, we need to allow actions to be planning actions—*i.e.*, it may be that the best thing to do is to set a subgoal and do some more planning before any further physical action.

3.4 Objectives

Our notion of **objective** is similar to some uses of the term "goal." But the term goal is used is different ways in the literature: goals are sometimes the intentions driving an agent's behavior, at other times they are the input to a planning process, and sometimes they are simply the main effects of a recipe. Goals are sometimes considered to be states of the world to attain (*e.g.*, the goal is a situation where block A is on block B), or sometimes an action that must be performed (*e.g.*, the goal is to open the door).

We will try to avoid all this ambiguity by not using the word goal any further. An **objective** is an intention that is driving our current behavior. Objectives are expressed in the form of abstract actions, such as winning the lottery, or getting block A onto block B. Objectives are not just any actions. They are actions that are defined in terms of their effects, and cannot be executed directly. To accomplish objectives, we need to choose or build a recipe that, if followed, leads to a state in which effects of the objective hold.

3.5 Resources

The final key concept in the abstract model is that of a **resource**. A resource is a object that is used during the execution of a recipe. Resources might be consumable (*i.e.*, cease to exist in their prior form) as a result of the recipe (*e.g.*, as ingredients are consumed when making a cake), or might be reusable (*e.g.*, as a hammer is used to drive in a nail). In a traditional planning model, resources are the objects that are used to bind the variables in the plan and, in fact, many applications of planning are essentially resource allocation problems.

4. PROBLEM SOLVING

Once we have the concepts defined in the last section, we can now give an quick overview of our model of a single agent's problem solving behavior. Just as task-level actions affect the state of the world, problem-solving actions affect the cognitive state of the agent, which we represent (for purposes of this paper) as the **problem solving (PS) state**. The problem solving state consists of the agents commitments towards objectives, the recipes for achieving those objectives, the resources used in those recipes, and so on.

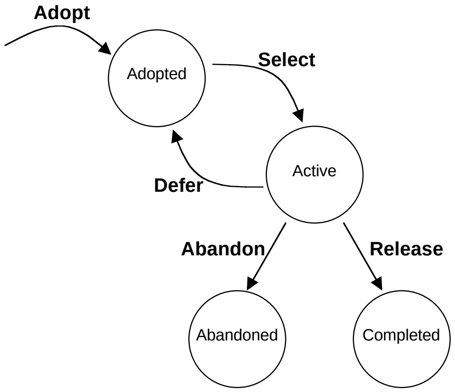


Figure 2: Life cycle of an intention

The PS state must contain at least the following information:

1. The current situation: what the agent believes (or assumes) to be true as a basis for acting, including what resources are available;
2. Intended objectives: a forest of objectives that the agent has adopted, although not all are necessarily motivating its current action. Each tree in the forest captures a subobjective hierarchy for a particular root objective;
3. Active objective(s): the intended objective(s) that is (are) currently motivating the agent's action; An objective tree that includes an active objective is an active objective tree;
4. Intended recipes: the recipes and resources that the agent has chosen for each of the intended objectives;
5. Recipe library: a set of recipes indexed by objective and situation types. The library need not be static and may be expanded by planning, learning, adaptation, *etc.*

An agent's problem solving activity involves exploring the current situation, adopting objectives, deciding on courses of action and resources to use, performing actions, and using the results of actions to further modify its objectives and future actions. The problem solving actions (see Figure 1) are divided into two classes, one concerned with intention and commitment and one concerned with knowledge and reasoning.

4.1 PS Acts Relating to Commitment

In our model of agent behavior (similar to [7, 15]), an agent is driven by its intentions, in the form of objectives, recipes, and resource uses to which it is committed. Intentions move through a life cycle shown in Figure 2.

In order to act, an agent must form intentions by means of a commitment act that we call **Adopt**. For example, an agent might adopt a particular recipe for a certain objective, say to plan a trip by using a recipe to call a travel agent. If they change their mind, they may drop the commitment using a act we call **Abandon**. For instance, the agent may change their mind, abandon the recipe to call the travel agent and adopt a recipe to book a ticket on the web. Similarly, an agent may adopt or abandon objectives, and adopt and abandon commitments to use certain resources.

An agent may have several different objectives that it is committed to, and even with respect to one objective, there may be several sub-objectives that could be chosen to drive the agent's action. The action of choosing the objective(s) to motivate the next behavior is called **Select**. Once an objective is selected, the agent may perform reasoning to elaborate on its associated recipe, or to evaluate an action that that recipe suggests, and may eventually select an action to perform. If an agent's priorities change, it may **Defer** the objective or action, leaving it as an intention to be addressed later. Finally, when an agent believes that an objective has been achieved, it may **Release** the objective, thereby removing it from its set of intentions.

4.2 PS Acts Relating to Reasoning

Before committing, an agent will typically perform some reasoning. One of the key operations is to determine what options are available, which we call **Identify**. For example, an agent may identify a possible recipe for achieving some objective, or identify certain resources that are available. They may even identify possible goals to pursue and consider them before making any commitment. Once an option is identified, the agent may **Evaluate** it relative to its purpose. For instance, it might evaluate a recipe to see how well it might accomplish its associated objective, or evaluate an objective to see if it is worthwhile, or evaluate a resource or action to see how well it serves a particular recipe. In addition, an agent may choose to **Modify** a certain objective, recipe or resource to produce a another that then could be evaluated.

In addition to reasoning about possible goals and actions, an agent may also reason about its current situation. Situations may be identified by exploring them further, and may be evaluated to see how desirable the current (or expected) situation is and whether it should plan to change it. Agents that act and do little planning would only care about the current situation they are in, and all activity would be tied to that situation. More complex agents, however, could do planning in hypothetical situations, or want to act based on certain assumptions.

4.3 Problem Solving Behavior

With these elements of the problem solving model in place, we can describe how an agent solves problems. It is convenient to present this activity as occurring in a series of phases. In practice, an agent may short circuit phases, or return to prior phases to reconsider their commitments at any time.

1. *Determining the Objective:* An agent may at any time reconsider the objectives it has, adopt new ones, abandon old ones, and otherwise modify and adjust them. Of course effective agents will not spend too much time reconsidering and evaluating their objectives, but will spend their effort in pursuing an objective. To do this, they must first select one or more objectives to pursue. These are the active objectives.

2. *Determining the Recipe:* Given an active objective, an agent must then determine a recipe to follow that may achieve the objective. It may be that a recipe has already been used for some time to determine the agent's actions in pursuing the objective, and the agent may simply invoke the recipe once again in the current situation to determine what to do next. But the agent might also consider switching to another recipe, refining an existing recipe, or actually building a new recipe for this objective. In these latter cases, the next action the agent does is a planning action that results in a modified (or new) recipe for the objective.

3. *Using the Selected Recipe:* Given a selected recipe, the agent can then identify the next action to perform. If the recipe returns a sub-objective, then the agent needs to restart the process of evaluating objectives and choosing or constructing recipes. If the recipe indicates an atomic action, the agent can evaluate the desirability of the proposed action, and if it seems reasonable, perform the action. At that point, the situation has changed and the process starts again.

To implement such a problem solving agent, we would need to specify strategies for when objectives, recipes and proposed actions are evaluated and reconsidered, versus how often the current objective, recipe or proposed action is just taken without consideration. Agents that performed more evaluation and deliberation would be more careful and might be able to react better to changing situations, whereas agents that do less evaluation would probably be more responsive but also more brittle. The specifics of these strategies are not the focus of this paper.

5. COLLABORATIVE PROBLEM SOLVING

We now turn to the central issue of collaborative problem solving. When two agents collaborate to achieve goals, they must coordinate their individual actions. To mirror the development at the problem solving level, the collaborative problem solving level (see Figure 1) operates on the collaborative problem solving (CPS) state, which captures the joint objectives, the recipes jointly chosen to achieve those objectives, the resources jointly chosen for the recipes, and so on.

The collaborative problem solving model must serve two critical purposes. First it must provide the structure that enables and drives the interactions between the agents as they decide on joint objectives, actions and behavior. In so doing, it provides the framework for intention recognition, and it provides the constraints that force agents to interact in ways that maintain the collaborative problem solving state. Second, it must provide the connection between the joint intentions and the individual actions that an agent performs as part of the joint plan, while still allowing an agent to have other individual objectives of its own.

While we talk of shared objectives, intended actions and resources, we do not want to require that agents have the same library of recipes to choose from. This seems too strong a constraint to place on autonomous agents. We assume only that the agents mutually agree on the meaning of expressions that describe goals and actions. For example, they might both understand what the action of taking a trip entails. The specific recipes each has to accomplish this action, however, may be quite different. Their recipes may accomplish subgoals in different orders for instance (one may book a hotel first, then get a air ticket, where the other might reverse the order). They might break the task down into different subgoals (*e.g.*, one may call a travel agent and book flight and hotel simultaneously, while the other might book flights with an agent and find hotels on the web). And for any subgoal, they might pick different actions (*e.g.*, one might choose a flight that minimizes cost, whereas the other might minimize travel time). To collaborate, the agents must agree to some level of detail on a new abstract joint recipe that both can live with. The joint recipe needs be refined no further in places where the two agents agree that one agent is responsible for achieving a sub-objective.

Establishing part of the collaborative problem solving state requires an agreement between the agents. One agent will propose an objective, recipe, or resource, and the other can accept , reject or produce a counterproposal or request further information. This is the level that captures the agent interactions. To communicate, the agent receiving a message must be able to identify what CPS act was intended, and then generates responses that are appropriate to that intention. In agent-communication languages between programs, the collaborative act would be explicit. In human-agent communication based on natural language, a complex intention recognition process may be required to map the interaction to the intended CPS act. This will be described in further detail in the Interaction section below, after the abstract collaborative model is described.

5.1 Collaborative Problem Solving Acts

As a first cut, the collaborative problem solving level looks just like the PS level, except that all acts are joint between the collaborating agents. We will name these CPS acts using a convention that just applies a prefix of "c-". Thus the *c-adopt-objective* act is the action of the agents jointly adopting a joint objective.

While we can model an individual agent adopting an individual objective as a primitive act in our model at the PS level, there is no corresponding primitive act for two agents jointly adopting a goal. This would require some sort of mind synchronization that it not possible. We agree with researchers such as Grosz and Sidner [8] and Cohen and Levesque [3] in that joint actions must be composed out of individual actions. There remains a meaningful level of analysis that corresponds to the PS level model if we view the CPS acts as complex acts, *i.e.*, objectives, that the agents recognize and use to coordinate their individual actions. The constraints on rational behavior that an agent uses at the PS level have their correlates at the collaborative PS level, and these inform the intention recognition and planning behavior of the agents as they coordinate their activities. For instance, a rational individual agent would not form an objective to accomplish some state if it believed that the state currently holds (or will hold in the future at the desired time). Likewise, collaborating individual agents would not form a collaborative objective to achieve a state that they jointly believe will hold at the (jointly) desired time. The analysis of the behavior at this abstract level provides a simple and intuitive set of constraints on behavior that would be hard to express at the interaction action level.

5.2 The Interaction Level

The interaction level provides the connection between the communicative acts (*i.e.*, speech acts) that the agents perform, such as requesting, informing, warning, and promising, and the collaborative problem solving acts they jointly perform. In other words, it deals with the individual actions that agents perform in order to engage in collaborative problem solving. All the acts at this level take the form of some operator applying to some CPS act. For instance, an agent can **Initiate** a collaborative act by making a proposal and the other agent can **Complete** the act (by accepting it) or **Reject** it (in which case the CPS act fails because of lack of "buy in" by the other agent). In more complex interactions, an agent may **Continue** a CPS act by performing clarification requests, elaborations, modifications or counter-proposals. The interaction-level acts we propose here are similar to Traum's [17] grounding act model, which is not surprising as grounding is also a form of collaborative action.

From a single agent's perspective, when it is performing an interaction act (say, initiating adoption of an joint objective), it must plan some communicative act (say, suggesting to the other agent that it be done) and then perform (or *realize*) it. On the other side of the coin, when one agent performs a communicative act, the other agent must *recognize* what interaction act was intended by the performer. Identifying the intended interaction acts is a critical part of the intention recognition process, and is essential if the agents are to maintain the collaborative problem-solving state. For instance, consider a kitchen design domain in which two agents collaborative to design and build a kitchen. The utterance "Can we put a gas stove beside the refrigerator" could be said in order to (1) ask a general question about acceptable practice in kitchen design; (2) propose adding a stove to the current design; or (3) propose modifying the current design (say by using a gas stove rather than an electric one). Each of these interpretations requires a very different response from the hearer and, more importantly, results in a different situation for interpreting all subsequent utterances. Each one of these interpretations corresponds to a different collaborative problem solving act. If we can identify the correct act, we then have a chance of responding appropriately and maintaining the correct context for subsequent utterances.

We should note that the interaction level is not just required for natural language interaction. In other modalities, the same processes must occur (for example, the user initiates a joint action by clicking a button, and the system completes it by computing and displaying a value). In standard agent communication languages, these interaction level actions are generally explicit in the messages exchanged between agents, thereby eliminating the need to recognize them (although not to the need to understand and perform them oneself).

5.3 Examples

To put this all together, consider some typical but constructed examples of interactions. These examples are motivated by interactions we have observed in a medical advisor domain in which the system acts to help a person manage their medications. These examples are meant to fit together to form a constructed dialogue that illustrates a number of points about the CPS level analysis.

The simplest collaborative acts consist of an initiate-complete pair. For example, here is a simple *c-identify* of a *situation*:

U: Where are my pills? (1)
S: In the kitchen (2)

Utterance (1) is a Wh-question that initiates the *c-identify-situation* act, and utterance (2) answers the question and completes the CPS act.[2] When utterance (2) is done, the two agents will have jointly performed the *c-identify-situation* action.

Utterances may introduce multiple collaborative acts at one time, and these may be completed by different acts. For instance:

S: It's time to take an aspirin (3)
U: Okay (4)
U: [Takes the aspirin] (5)

Utterance (3) is a suggestion that U take an aspirin, which initiates both a *c-adopt-objective* (to intend to take medication currently due) and a *c-select-action* (to take an aspirin). Utterance (4) completes the *c-adopt* action and establishes the joint objective. Action (5) completes the *c-select* action by means of U performing the PS-level act *select* on the action, resulting in the action being performed.

Many more complex interactions are possible as well. For instance:

U: What should we do now? (6)
S: Let's plan your medication for the day (7)
U: Okay (8)

Utterance (6) is a question that initiates a *c-adopt-objective*, utterance (7) continues this act by answering the question with a suggestion, and utterance (8) completes the act (thus establishing the joint objective). Note that the objective agreed upon is itself a collaborative problem solving act—they have established a joint objective to perform a *c-adopt* action for some as yet unspecified *recipe*. This could then lead to pursuing a sub-objective such as creating a recipe as in the following interaction:

S: You could take your celebrex at noon. (9)
U: Will that interfere with my lunch date (10)
S: No. (11)
U: OK. I'll do that (12)

Utterance (9) is a suggestion that initiates a *c-identify-recipe* and continues the previously established *c-adopt-objective* action. Utterance (10) completes the *c-identify* of the recipe (by grounding the suggestion), continues the *c-adopt* action, and initiates a *c-evaluate* of the recipe by exploring a possible problem with the suggested action. Utterance (11) completes the *c-evaluate* act by answering the question, and utterance (13) then completes the *c-adopt* act by agreeing to the recipe initially suggested in (9).

6. EXTENDED EXAMPLE

To better illustrate the complexity of even fairly simple collaborative problem solving, the following is an example of a session with a prototype Medication Advisor system under development at Rochester [5]. The Medication Advisor is designed to help people manage their prescription medication regimes—a serious real-world problem that has a significant impact on people's health.

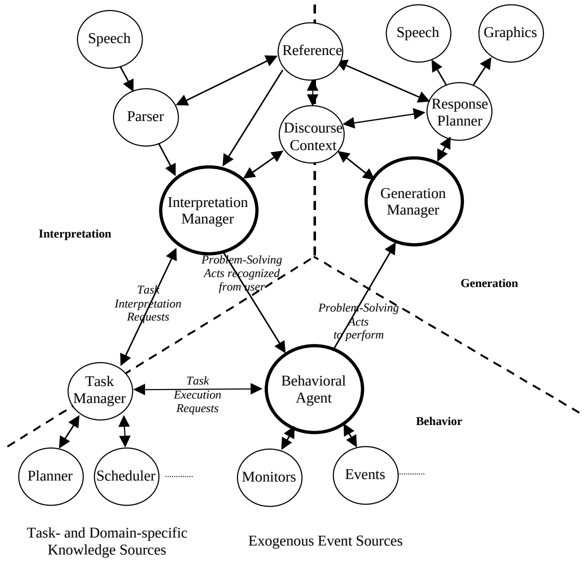


Figure 3: TRIPS collaborative system architecture (from [1])

To follow the problem solving, we need to understand something of the architecture of the system. The Medication Advisor is an application of the TRIPS spoken dialogue system [6], whose architecture is shown in Figure 3. As described in [1], the main components of the system as regards problem solving are as follows:

- The Interpretation Manager (IM), which maintains the discourse context and recognizes user intention from their utterances;
- The Behavioral Agent (BA), which manages system problem solving obligations and drives system behavior;
- The Task Manager (TM), which maintains the collaborative problem solving state and supports both interpretation and execution with task- and domain-specific knowledge; and
- The Generation Manager (GM), which coordinates the generation of spoken speech and multimodal output, among other functions.

This system is a fully implemented, end-to-end speech-driven dialogue system, although for this session some aspects of the problem-solving model were not fully implemented. For brevity, we have suppressed most of the detail of the actual message traffic between components of the system, focusing solely on the interactions within the collaborative problem solving model. Utterances are labeled by the agent and a number indicating the sequence of speaker turns.

The user starts by asking a question:

U: Can I take an aspirin? (1)

The Interpretation Manager asks the Task Manager to interpret the utterance in the current problem solving context. The Task Manager replies that the utterance looks like an *initiate* of a *c-evaluate-action*. That is, the user

[2] Note that we are ignoring grounding issues in this paper. In a dialogue system, the CPS act is not actually completed until the answer to the question is grounded by U, say by the utterance such as "OK" or "thanks".

seems to be initiating a collaborative evaluation of the action of taking aspirin. The Interpretation Manager uses this result, together with other interpretation steps, and broadcasts that as the system's understanding of what the user has just done.

When the Task Manager receives this message, it can update its state to include the fact that, yes, the user has initiated a new collaborative act. Meanwhile, the Behavioral Agent also receives the notification. It decides, based on its current obligations, that the system should attempt to complete the action. To do this, it asks the Task Manager to perform the (non-collaborative) act of evaluating an action. The Task Manager does this by following a recipe for evaluating that type of action, which involves querying databases for what medications the user is taking and whether any of those interact with aspirin. It turns out that the there is an interaction, so the evaluation is negative. The Behavioral Agent, upon receipt of the result of the evaluation, requests that the Generation Manager perform the communicative act *inform* (with the evaluation results) in order to realize the interaction act of completing the user's *c-evaluate-action* CPS act.

The Generation Manager determines what to say:

S: No, you are taking celebrex and celebrex interacts with aspirin. (2)

Once the speech has been generated, the Generation Manager can broadcast to the rest of the system that the *inform* act is done. Note that the interaction act (the *complete*) is not yet done, since this depends on the user having understood the system's utterance (*i.e.*, on its being grounded).

The user responds with:

U: Oh. (3)

The Interpretation Manager takes this as grounding its utterance (2), since it does not indicate any lack of understanding. This fact is broadcast to the rest of the system. Once the system's utterance is grounded, the Generation Manager can announce that the interaction act is done. As there is no additional problem solving content in the user's utterance (3), the system does not do anything in response.

The user continues with:

U: Could Celebrex cause headaches? (4)

As with (1), the IM asks the TM to interpret the utterance in the current problem solving state. Note that the IM must explicitly indicate that this interpretation should be relative to the context in which the system's interaction act (the *complete* of the *c-evaluate-action*) is done. This is just one example of the explicit synchronization necessary in order to implement the collaborative problem solving model in a distributed system.

The TM answers that the utterance (4) looks like an *initiate* of a *c-identify-situation* (as to whether celebrex causes headaches). With this choice of an interpretation, the Task Manager can update its model of what the user is doing. Meanwhile, the Behavioral Agent decides that the system should perform its part of the collaborative action in order to complete it. The TM does the *identify-situation* and responds that it is not the case that celebrex can cause headaches. The BA passes this answer to the GM, requesting that it inform the user in order to complete the collaborative act. This results in the following system utterance:

S: No, headaches are not an expected side-effect of celebrex. (5)

And again, the *inform* is done once the speech has been produced.

Meanwhile, the TM, in the process of updating its state based on the user having initiated the *c-identify-situation* regarding celebrex causing headache, has performed some plan recognition and thinks it is likely that the user may be trying to cure their headache. Note that up to this point, the user has not said anything about having a headache—this is purely an inference based on task and domain knowledge. The TM reports that this is a problem which should be resolved, although it leaves it to the BA to prioritize the system's objectives and decide what to do.

In this case, the BA decides to take initiative and requests that the TM suggest what to do. The TM responds that the system should initiate a *c-identify-situation* regarding whether the user has a headache. The BA sends this to the GM, resulting the following utterance:

S: Do you have a headache? (6)

Once the speech has been output, the GM announces that the *ask* is done. At this point both interaction acts ("headaches are not a side-effect of celebrex" and "do you have a headache") are awaiting grounding, and the question is awaiting a response from the user. When the user answers with:

U: Yes. (7)

Both system utterances (5) and (6) are grounded, so both pending interactions acts are marked as completed, and the system proceeds to interpret the user's utterance (7) in the resulting context.

The dialogue for another continues for another fifteen utterances as the system addresses the user's headache and then supports them with several other aspects of their medication regime. Unfortunately, space precludes an extended presentation.

7. RELATED WORK

As noted above, work has been done on general theoretical frameworks for collaborative interaction [3, 9, 11]. However, the focus of these models was more specifying the mental details (beliefs, intentions, etc.) of such collaboration, whereas the focus of our model is describing practical dialogues. Also, in these proposals, many details of what the models would look like are not given. The SharedPlans formalism [9, 11], for example, *does* expressly model the adoption of recipes (Select_Recipe, Select_Recipe_GR), but that is as far as it goes. We believe that our model will prove to be complementary to this formalism, with the remainder of problem solving acts either existing at some higher level (e.g. adopt/abandon/evaluate-objective), being added to the same recipe level (evaluate-recipe), or being part of the unspecified Elaborate_Individual/Elaborate_Group processes.

Our belief that human-machine interaction can occur most naturally when the machine understands and does problem solving in a similar way to humans is very close to the philosophy upon which the COLLAGEN project [16] is founded. COLLAGEN is built on the SharedPlan formalism and provides an artificial language, human-computer interface with a software agent. The agent collaborates with the human through both communication and observation of actions. COLLAGEN, as it works on a subset of the SharedPlans formalism, also does not explicitly model most of our problem solving acts.

Several dialogue systems have divided intention recognition into several different layers, although these layerings are at much different levels than our own. Ramshaw [14] analyzes intentions on three levels: domain, exploration, and discourse. Domain level actions are similar to our own domain level. The discourse level deals with communicative actions. The exploration level supports a limited amount of evaluations of actions and plans. These, however, cannot be directly used to actually build up a collaborative plan, as they are on a stack and must be popped before the domain plan is added to.

Lambert and Carberry [10] also had a three level model, consisting of domain, problem solving, and discourse levels. Their problem solving level was fairly underdeveloped, but consists of such recipes as *Build_Plan* and *Compare_Recipe_by_Feature* (which allow the comparison of two recipes on one of their features). The model does not include other of our problem solving acts, nor does it explicitly model collaboration, interaction acts, etc.

These models assumed a *master-slave collaboration paradigm*, where an agent must automatically accept any proposal from the other agent. Chu-Carroll and Carberry [2] extended the work of Lambert and Carberry, adding a level of proposal and acceptance, which overcame the master-slave problem. However, Chu-Carroll and Carberry (along with Ramshaw and Lamber and Carberry), assume a shared, previously-specified problem solving *plan* which is being executed by the agents in order to collaborate. This restricts collaboration to homogeneous agents which have identical problem solving plans, whereas in our model, there is no set problem solving plan, allowing agents with different individual problem solving strategies to collaborate.

Finally, Elzer [4] specifically mentions the need for a problem-solving model in discourse, citing dialogue segments similar to those that we give. However, she offers no proposal of a solution.

8. CONCLUSIONS

The collaborative problem solving model presented in this paper offers a concrete proposal for modeling collaboration between agents, including in particular between human and software agents. Our model is based on our experience building collaborative systems in several problem solving domains. It incorporates as many elements as possible from formal models of collaboration, but is also driven by the practical needs of an implemented system.

9. ACKNOWLEDGMENTS

This material is based upon work supported by Dept. of Education (GAANN) grant no. P200A000306; ONR research grant no. N00014-01-1-1015; DARPA research grant no. F30602-98-2-0133; NSF grant no. EIA-0080124; and a grant from the W. M. Keck Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above-mentioned organizations.

10. REFERENCES

[1] J. Allen, G. Ferguson, and A. Stent. An architecture for more realistic conversational systems. In *Proceedings of IUI-2001*, Santa Fe, NM, January 14-17 2001.

[2] J. Chu-Carroll and S. Carberry. Conflict resolution in collaborative planning dialogues. *International Journal of Human-Computer Studies*, 53(6):969–1015, 2000.

[3] P. Cohen and H. Levesque. Intention is choice with commitment. *Artifical Intelligence*, 42:213–261, 1990.

[4] S. Elzer. The role of user preferences and problem-solving knowledge in plan recognition for expert consultation systems. In *Working Notes of the IJCAI-95 Workshop on The Next Generation of Plan Recognition Systems*, pages 37–41, Montreal, Canada, 1995.

[5] G. Ferguson, J. Allen, N. Blaylock, D. Byron, N. Chambers, M. Dzikovska, L. Galescu, X. Shen, R. Swier, and M. Swift. The Medication Advisor project: Preliminary report. Technical Report 776, CS Dept., U. Rochester, May 2002.

[6] G. Ferguson and J.F. Allen. TRIPS: An integrated intelligent problem-solving assistant. In *Proceedings of AAAI-98*, pages 567–573, Madison, WI, 28–30 July 1998.

[7] M.P. Georgeff. Actions, processes, and causality. In *Proceedings of the Workshop on Reasoning about Actions and Plans*, Los Altos, CA, 30 June–2 July 1986.

[8] B. Grosz and C. Sidner. Attention, intention, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[9] B.J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[10] L. Lambert and S. Carberry. A tripartite plan-based model of dialogue. In *Proceedings of ACL-91*, pages 47–54, Berkeley, CA, June 1991.

[11] K.E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 24(4):525–572, 1998.

[12] J. McCarthy. First order theories of individual concepts and propositions. In J.E. Hayes, D. Michie, and L.I. Mikulich, editors, *Machine Intelligence*, volume 9, pages 129–174. Ellis Horwood, Chichester, England, 1979.

[13] M.E. Pollack. The uses of plans. *Artificial Intelligence*, 57, 1992.

[14] L.A. Ramshaw. A three-level model for plan exploration. In *Proceedings of ACL-91*, pages 39–46, Berkeley, CA, June 1991.

[15] A.S. Rao and M.P. Georgeff. An abstract architecture for rational agents. In *Proceedings of KR-92*, pages 439–449, Boston, MA, 25–29 October 1992.

[16] C. Rich and C.L. Sidner. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3–4):315–350, 1998.

[17] D.R. Traum. *A Computational Theory of Grounding in Natural Language Conversation.* PhD thesis, CS Department, U. Rochester, December 1994.